IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received April 21, 2021, accepted May 8, 2021, date of publication May 13, 2021, date of current version May 25, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3080176

Rotary Inverted Pendulum Identification for Control by Paraconsistent Neural Network

ARNALDO DE CARVALHO, JR.[1,2], JOÃO FRANCISCO JUSTO[2], BRUNO AUGUSTO ANGÉLICO[2], ALEXANDRE MANIÇOBA DE OLIVEIRA[1], (Member, IEEE), AND JOÃO INÁCIO DA SILVA FILHO[3], (Member, IEEE)

[1]Federal Institute of Education Science and Technology of Sao Paulo (IFSP), Cubatão 11533-160, Brazil
[2]Escola Politécnica, Universidade de São Paulo, São Paulo 05508-010, Brazil
[3]Laboratory of Applied Paraconsistent Logic, Department of Electronic Engineering and Computation, Santa Cecília University (UNISANTA), Santos 11045-907, Brazil

Corresponding author: Arnaldo de Carvalho, Jr. (adecarvalhojr@ifsp.edu.br)

This work was supported by Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brazil (CAPES) - Finance Code 001.

ABSTRACT Artificial neural networks (ANNs) have been used over the last few decades to perform tasks by learning with comparisons. Fitting input-output models, system identification, control, and pattern recognition are some fields for ANN applications. However, problems involving uncertain situations could be challenging for them. The family of paraconsistent logics (PL) is a powerful tool that can deal with uncertainty and contradictory information, so getting attention from researchers for its implications and applications in artificial intelligence. This investigation describes a novel activation function reasoned on the paraconsistent annotated logic by two-value annotations (PAL2v) rules, a variation of PL, allowing the design of a new paraconsistent neural net (PNN), applied in model identification for control (I4C) of a closed-loop rotary inverted pendulum (RIP) system.

INDEX TERMS Paraconsistent logic, neural net, model identification, pattern analysis, rotary inverted pendulum.

I. INTRODUCTION

Artificial neural networks are mathematical algorithms that can learn a specific function or pattern [1], [2], could be applied in a wide range of applications [3]. Because of those characteristics, the ANN is a powerful tool for identification and system control [1], [2]. In recent years, the ANN application for identification and control has been investigated, for online and offline environments [4], [5]. The model identification approach oriented to control a system, when the model is valid only in the boundary conditions of this purpose, is called identification for control (I4C) [6]–[8].

Noise, disturbances, and uncertainties are inherent aspects of real-world problem description [9], not usually considered by the state estimation methods for nonlinear control techniques [10]. The ANN attractiveness in identification and control derives from its intrinsic ability to use experimental data to model unknown systems, even with perturbations or uncertainties, enabling a notable alternative in situations where conventional methods could fail in determining the appropriate control laws [11].

Uncertainty, inconsistency, and contradictory signals are also aspects considered by the paraconsistent logic (PL), which deals with them without resulting in triviality [12], [13]. PL is a family of alternative logics that repeals the principle of non-contradiction of classical logic. PL attracted interest not only philosophical but by its implications and applications in artificial intelligence too [14]. Through the use of pieces of evidence, the PL can model human knowledge and reasoning, allowing applications in specialist systems that consider uncertain information in decision-making [13], [15], [16].

The Paraconsistent Annotated Logic with 2-value annotations (PAL2v), belonging to the PL family, uses two variables, or evidence, which allows greater representation power in a lattice of the Real plane, when expressing knowledge about a proposition P [15], [17]. PAL2v allows an expert system to deal with concepts like uncertain, inconsistent, and incomplete data, such as those obtained from sensors, without the risk of trivialization [17], [18]. A hybrid proportional-integral (PI) control approach, using PAL2v cells to handle the

The associate editor coordinating the review of this manuscript and approving it for publication was Shen Yin.

controller inputs, has been implemented in process control with excellent results [9].

Here, we propose a novel paraconsistent neural net (PNN), built with activation functions reasoned on the PAL2v equations and rules. We describe the respective algorithm and its application on the model identification of a rotary inverted pendulum (RIP) system stabilized by closed-loop control. Researchers consider the inverted pendulum in modeling and control research due to its wide range of applications, including aerospace and robotics [19]. Besides this introduction, we organized the text as follows: Section II presents the related work and theoretical background for ANN and PAL2v. In Section III, we describe the novel PAL2v neuron and the mathematical relative to the proposed method. The experimental results and comparison with other activation functions. Here, we propose a novel paraconsistent neural net (PNN), built with activation functions reasoned on the PAL2v equations and rules. We describe the respective algorithm and its application on the model identification of a rotary inverted pendulum (RIP) system stabilized by closed-loop control. Researchers consider the inverted pendulum in modeling and control research due to its wide range of applications, including aerospace and robotics [19]. Besides this introduction, we organized the text as follows: Section II presents the related work and theoretical background for ANN and PAL2v. In Section III, we describe the novel PAL2v neuron and the mathematical relative to the proposed method. The experimental results and comparison with other activation functions, including Leaky ReLU [20], are given in Section IV. Section V concludes with considerations about the paraconsistent neural network based on results presented in Section IV., including Leaky ReLU [20], are given in Section IV. Section V concludes with considerations about the paraconsistent neural network based on results presented in Section IV.

II. RELATED WORK

This section provides the theoretical background for the proposed paraconsistent neural net algorithm, divided into two: ANN architecture and paraconsistent logic.

A. ARTIFICIAL NEURAL NETWORK OVERVIEW

Artificial neurons have mathematical functions inspired by biological neurons and form the basis of the ANNs. The possibility of describing their building blocks by simple computational devices and the network function determined by connections between neurons are two similarities between the artificial and biological neural networks [1]. Fig. 1 presents the k-th artificial neuron of a neural network. The artificial neuron can be described by Eq. (1)

$$a = f(n) = f(\mathbf{w}^\top \mathbf{p} + b), \tag{1}$$

where $\mathbf{p} = \begin{bmatrix} p_1 & p_2 & \dots p_R \end{bmatrix}^\top$ is the input vector, a is the output, $\mathbf{w} = \begin{bmatrix} w_{k,1} & w_{k,2} & \dots w_{k,R} \end{bmatrix}^\top$ is the weighting vector and b is a bias correction [1].

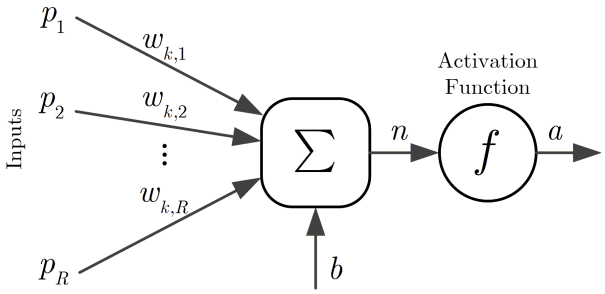


FIGURE 1. Multiple-Input Neuron.

The combination of weights, bias, and activation function allows the neuron to seek for approximating a target function. The activation function must be derivable, allowing the weight and bias calibration by regression algorithms, such as the stochastic gradient descent applied in the feed-forward or back-propagation learning algorithms [1].

The interconnection of several neurons in layers form artificial neural networks (ANN) [1], [2].

The literature presents different activation functions, such as rigid, linear limit, log-sigmoid (sigmoid), tangent hyperbolic sigmoid (tanh), positive linear (poslin) [1], [21], also called rectified linear unit (ReLU), Leaky-ReLU (LReLU), among others [1], [3], [20]. The sigmoid is one of the most used activation functions until recently [3]. The sigmoid (σ) is given by (2) [21].

$$a = \sigma(n) = \frac{1}{1 + e^{-n}}, \tag{2}$$

and its derivative is

$$\sigma'(n) = \sigma(n)(1 - \sigma(n)). \tag{3}$$

The output varies from 0 (deactivation) to 1 (activation). Its derivative tends to 0 for input values above +5 and below −5.

The tanh is another sigmoid-like function, varying from −1 to +1, whose expression and derivative are given by (4) and (5), respectively.

$$a = \tanh(n) = \frac{e^n - e^{-n}}{e^n + e^{-n}}, \tag{4}$$

$$\tanh'(n) = 1 - \tanh^2(n). \tag{5}$$

ReLU is very similar to the identity function, making the neural net learning process, based on this activation function, faster than sigmoid [20]. The ReLU function and its derivative are expressed as:

$$a = \mathrm{ReLU}(n) = \begin{cases} 0 & n \leq 0 \\ n & n > 0, \end{cases} \tag{6}$$

$$\mathrm{ReLU}'(n) = \begin{cases} 0 & n \leq 0 \\ 1 & n > 0. \end{cases} \tag{7}$$

A disadvantage of ReLU is that neurons tend to "die" during training, causing the neuron output to start producing only

zeroes. A variation of ReLU, called Leaky-ReLU (LReLU), avoid this [20], whose function and derivative are presented in (8) and (9).

$$a = \mathrm{LReLU}(n) = \begin{cases} \alpha n & n \leq 0 \\ n & n > 0, \end{cases} \tag{8}$$

$$\mathrm{ReLU}'(n) = \begin{cases} \alpha & n \leq 0 \\ 1 & n > 0, \end{cases} \tag{9}$$

where α is a parameter introduced in LReLU, with values proposed between 0.01 and 0.2 [20]. LReLU allows the unit to give a small gradient when the unit is no active ($n < 0$), reducing the potential problem mentioned about ReLU [21].

A robustness analysis with neural networks in control systems using tanh and ReLU nonlinear functions can be seen in [22].

In this study, a novel activation function reasoned on PAL2v rules is presented, preserving some characteristics of sigmoid and ReLU.

The following section presents the application of ANN in system identification, a relevant step towards the development of ANN-based controllers.

B. ANN IN IDENTIFICATION FOR CONTROL

In control system applications, the goal is generally to achieve good performance in a closed-loop. If the system identification is for control purposes, the goal is not to obtain the best possible model that fits the data, as in the classic system identification approach, but to get a model good enough to design a controller that results in good closed-loop performance instead. This approach is called identification for control (I4C) [7].

Many research and methodologies for identifying systems and develop control algorithms use inverted pendulum structures [23], [24]. Here, we considered a rotary inverted pendulum (RIP), also called Furuta and Iwase [25], as a system to be identified. The RIP consists of a driven arm that rotates in the horizontal (H) plane. Attached to the arm is a free pendulum to swing in the vertical (V) plane [26]. The reason is that the RIP carries many interesting features, such as nonlinearity, fast dynamics, multiple variables, and multidimensional spatial motion, allowing evaluation of several elements of modern control theory [26]–[28].

There are several investigations in the literature with the application of ANN to model a fast dynamic, unstable system, such as the RIP [29], also in closed-loop [30].

By collecting the inputs and outputs of the system, we can train the ANN, as represented in Fig. 2 [31].

Here, the system (RIP) is in closed-loop control, being u the signal from the controller, and y the RIP output. So, we can compare the system and ANN outputs. The estimated model output by ANN is $\hat{y}$. The weights update (w) to correct the ANN comes from the difference between y and $\hat{y}$. This process is repeated until the error reaches a minimum value. This procedure allows the ANN to generate nearly the closest response as the system for the same dataset applied [1], [2].

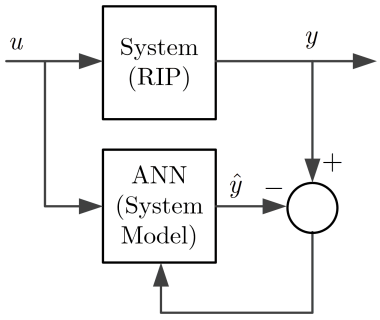


FIGURE 2. System (RIP) model identification.

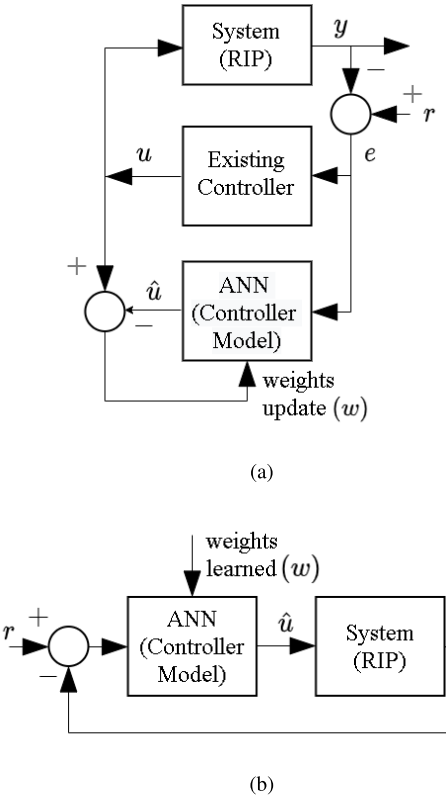


FIGURE 3. ANN learning the controller function (a). ANN taking control after the learning process (b).

An ANN can also learn, in a supervised manner, to behave as an existing control of the plant, as shown in Fig. 3a. The error (e) signal, from the difference between the reference (r) and y, is the input applied to the controller and ANN. In this case, the weights update (w) to correct the ANN comes from the difference between u and $\hat{u}$.

When the ANN learning process is complete, we can remove the original controller and the ANN can control the plant with similar performance [32], as presented in Fig. 3b.

The advantage of replacing the static control with a machine learning control, such as an ANN, is the possibility of learning continuously, online, and adapting according to uncertainties or system variations, allowing the design of adaptive controllers [2], [30]. The direct and inverse system

identification, by ANN, allow control strategies based on models, such as observed-based control, internal model control (IMC), model predictive control (MPC), model reference control (MRC), among others [2], [11], [33], [34].

By operating with several layers of neurons in parallel, an ANN could be used to model a dynamic multiple-input multiple-output (MIMO) system [29].

The following section presents the theoretical background behind the PAL2v to design the paraconsistent neural net proposed here.

C. PAL2V PARACONSISTENT LOGIC

The classical logic takes on stringent and inflexible binary laws that do not deal with redundancy, inconsistencies, or incomplete data situations [12], [35].

PL is a family of non-classical logic that presents the revocation of the non-contradiction principle as the pillar of its theoretical foundation, allowing it to reason with inconsistent information in a controlled and discriminating way [17].

We call paraconsistent annotated logic (PAL) when we represent the evidence of a proposition P by annotation through a lattice of four vertices [16]. The four extreme logic states indicated at the vertices of the PAL lattice are True (t), False (F), Undetermined ($\perp$), and Inconsistent ($\top$) [9].

One, two, or n values can form the annotation in the PAL lattice. An ordered pair allows a better accuracy about annotation than only a single value when expressing knowledge about the proposition P [17]. In this case, PL is called paraconsistent annotated logic with annotation of two values (PAL2v), using two inputs (μ_1, μ_2) normalized in a range from 0 to 1 [15], [17]. The first input, μ or μ_1, represents the degree of favorable evidence. The complement of the second input (μ_2) is called the degree of unfavorable evidence (λ), calculated by (7).

Through transformations in a unitary quadrant of the Cartesian plane (UQCP) and some algebraic interpretations, as well explained in [15], the PAL2v can be developed by equations (10) to (17). Fig. 4 shows the Lattice diagram for PAL2v. Note that the inputs (μ, λ) are orthogonal to each other. We define $\lambda = 1 - \mu_2$. From Fig. 4 it is possible to observe the following relations

$$D_C = \mu - \lambda \tag{10}$$

$$D_{CT} = \mu + \lambda - 1 \tag{11}$$

The horizontal axis of the Lattice represents the degree of certainty (D_C) defined in (10), within the interval $[-1,\ 1]$. The vertical axis of the Lattice represents the degree of contradiction (D_{CT}) as presented in (11), also within the interval $[-1,\ 1]$.

The resulting evidence degree (μ_E), calculated by (12), corresponds to (D_C) normalized in the range of 0 to 1, the same way that the resulting contradiction degree (μ_{ECT}) is the D_{CT} also normalized, as shown in (13).

$$\mu_E = \frac{D_C + 1}{2} = \frac{\mu - \lambda + 1}{2} \tag{12}$$

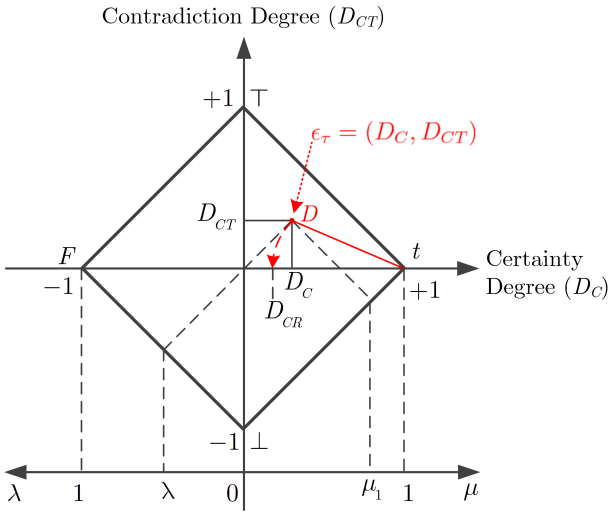


FIGURE 4. Lattice diagram of the PAL2v.

$$\mu_{ECT} = \frac{D_{CT} + 1}{2} = \frac{\mu + \lambda - 1 + 1}{2} = \frac{\mu + \lambda}{2} \tag{13}$$

The certainty interval (φ_E), defined as the range of certainty values in which the D_C can vary without being limited by D_{CT}, can be calculated by Eq. (14) [17].

$$\varphi_E = 1 - |D_{CT}| \tag{14}$$

By projecting the line segment D (15) over the D_C axis, where $d = \left(\left(1 - |D_C|\right)^2 + \left(D_{CT}\right)^2\right)^{0.5}$, we can extract the contradictory effects between both inputs, noting that D is within [0, 1]. Hence, the real certainty degree (D_{CR}) is found as (16) [15], [17].

$$D = \begin{cases} 1 & d \geq 1 \\ d & d < 1 \end{cases} \tag{15}$$

$$D_{CR} = \begin{cases} 1 - D & D_C > 0 \\ D - 1 & D_C < 0 \\ 0.5 & D_C = 0 \end{cases} \tag{16}$$

The resulting real evidence degree (μ_{ER}), in (17), corresponds to D_{C6R} normalized between 0 and 1, i.e.,

$$\mu_{ER} = \frac{D_{CR} + 1}{2} \tag{17}$$

Note that μ_E in (12) is linear, since it is a function of D_C. On the other hand μ_{ER} in (17) is nonlinear, because it depends on D.

The paraconsistent analysis node (PAN) [9], with its symbol shown in Fig. 5, is the core code for several paraconsistent cells presented in the literature [9], [17], [33].

III. METHODOLOGY

This session presents the proposed PAL2v neuron, the PNN description, including the respective mathematics.

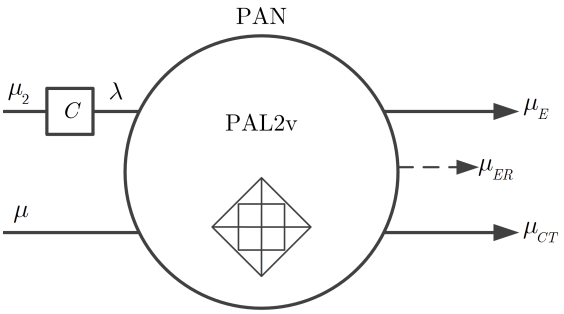


FIGURE 5. Symbol of PAN.

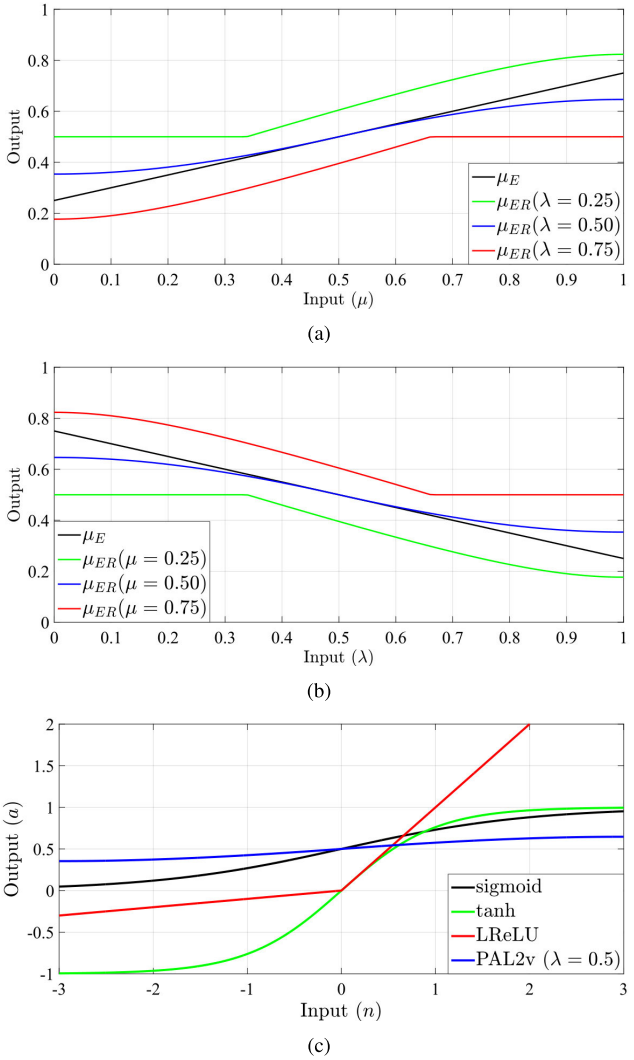


FIGURE 6. PAL2v neuron output μ_{ER} for λ fixed and μ variable (a). Output μ_{ER} for μ fixed and λ variable (b). Comparison between activation functions (c).

A. PARACONSISTENT NEURAL NETWORK

The novelty here is to use the PAL2v equations and rules as activation functions for neural networks. Fig. 6 shows the behavior of the μ_E and μ_{ER} outputs.

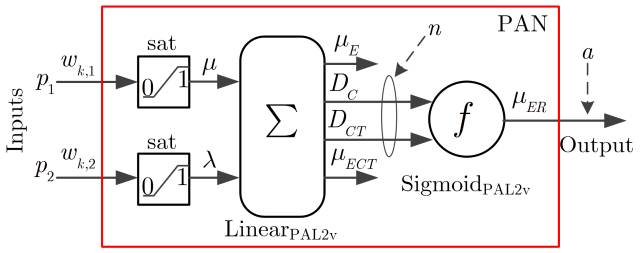


FIGURE 7. PAN as Activation Function of the PAL2v Neuron.

As μ_E output is linear, we call it $\text{linear}_{\text{PAL2v}}$ activation function. The output μ_{ER} is nonlinear. When one of the PAL2v neuron inputs is a constant equal to 0.5, μ_{ER} has a sigmoid shape. In this case, we can call the activation function $\text{sigmoid}_{\text{PAL2v}}$.

Fig. 6a shows the input μ varying from 0 to 1 and the input λ working as a "bias" with three fixed values. In Fig. 6b, the input μ is constant and λ varies within the range [0 1]. When one of the inputs (μ or λ) is at 0.5, the μ_{ER} has a sigmoid shape (blue line), with amplitude between 0.35 and 0.65. However, the green and red curves present a breaking point at 0.5. The reason is the limitation of D in 1 at maximum. The projection of D over the D_C axis cannot crossover the limit of the D_{CT} axis. When $D = 1$, D_{CR} is equal to 0, forcing μ_{ER} to be 0.5. The PAL2v neuron output, compared to other functions, is shown in Fig. 6c. The input n varies within $[-2 +2]$. Following the PAL2v rules, n is normalized between 0 and 1 and applied to the input μ, as in (18). Fig. 6c shows PAL2v output when λ is fixed at 0.5 (blue line).

$$\mu(x) = \frac{x - \min(x)}{\max(x) - \min(x)} \tag{18}$$

A single PAN can be understood as a neuron of 2 inputs, called PAL2v neuron, as presented in Fig. 7, where $\mu = \text{sat}(w_{1,1}\, p_1)$ and $\lambda = \text{sat}(w_{1,2}\, p_2)$, being sat the saturation function limited to the interval between 0 to 1.

It is possible to see the similarity between the PAL2v neuron and standard neuron presented previously in Fig. 1. Multi-input PAL2v neurons are possible, as Fig. 8.

In Fig. 8a, three PANs, with the linear output (μ_E) each, or "half" PAN, are connected to provide a pure 4-input PAL2v neuron, called $\text{summation}_{\text{PAL2v}}$. We added a "full" PAN to perform the neuron paraconsistent non-linear activation function (μ_{ER}). The resulting equations are:

$$\begin{aligned}
\mu_{E1} &= \frac{\mu_1 - \lambda_1 + 1}{2}; \\
\mu_{E2} &= \frac{\mu_2 - \lambda_2 + 1}{2}; \\
\mu_{E3} &= \frac{\mu_{E1} - \mu_{E2}}{2} = 0.25(\mu_1 - \lambda_1 - \mu_2 + \lambda_2) + 0.5
\end{aligned} \tag{19}$$

The equivalent simplified design is shown in Fig. 8b. In contrast to the neuron shown in Fig. 1, the bias of Fig. 8 now is part of the activation function inputs. If the bias is 0.5, a sigmoid-like shape is guaranteed.

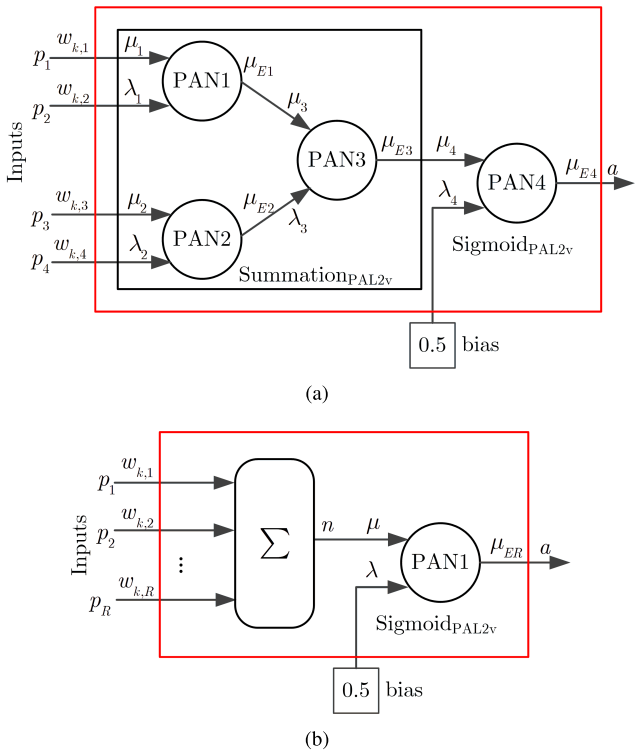


FIGURE 8. Pure 4-input PAL2v neuron (a), and the simplest multi-input PAL2v neuron (b).

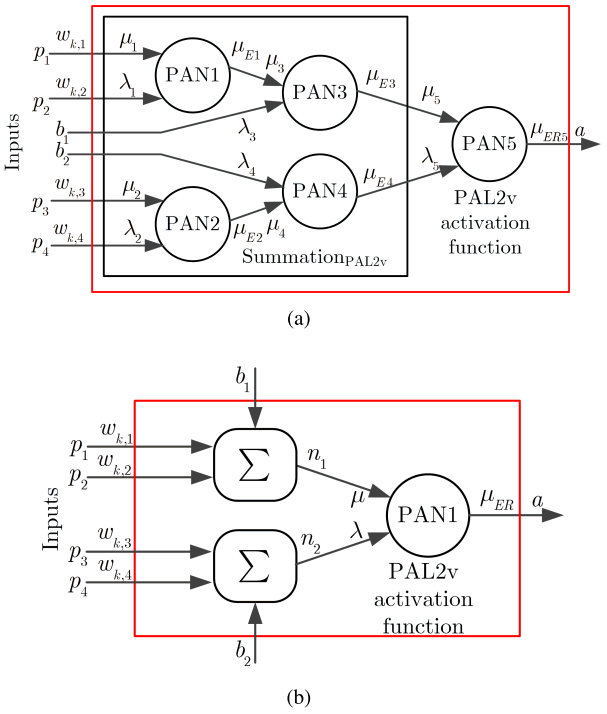


FIGURE 9. Alternative pure (a) and simplified (b) 4-input PAL2v neuron.

We show another configuration in Fig. 9. Here the bias is part of the summation instead of the sigmoid$_{PAL2v}$. In Fig. 9a, PAN1 and PAN3 form a 3-input summation$_{PAL2v}$ applied to μ_5 of PAN5, which performs the paraconsistent non-linear activation function of the neuron. PAN2 and PAN4 form another 3-input summation$_{PAL2v}$ connected to λ_5, such that:

$$\begin{aligned}\mu_5 &= \mu_{E3} = 0.25(\mu_1 - \lambda_1) - 0.5\lambda_3 + 0.75;\\ \lambda_5 &= \mu_{E4} = 0.25(\mu_2 - \lambda_2) - 0.5\lambda_4 + 0.75\end{aligned} \tag{20}$$

Now instead of being part of the PAL2v activation function, one bias is applied to each summation. In this case, the activation function is non-linear, but not necessarily a sigmoid, as presented in Fig. 6, since it depends of both, μ_5 and λ_5 values. Fig. 9b shows a simplified design.

By working with two orthogonal inputs, the PAL2v activation function offers more flexibility, as we can choose how to apply the signals between μ and λ and how to combine the PAN to configure the neurons.

Here, the common backpropagation and stochastic gradient descent are applied to calculate the level of adjusting required for each weight and bias [1], [33].

The chain rule is applied in the corresponding path inside the network. So we can calculate the impact of each weight and bias on the cost function (error) between the desired output and the PNN [1]. Therefore, the derivatives of each PAN node, relative to that particular path inside the PNN, are required. Following, we present the derivatives of the PAL2v non-linear neuron (μ_{ER}), relative to each input (μ, λ).

If μ is greater than λ, then D_C is positive. The derivatives are calculated as:

$$\begin{aligned}\mu_{ER} &= \frac{2 - \left(2\mu^2 + 2\lambda^2 - 4\mu + 2\right)^{0.5}}{2};\\ \frac{\partial \mu_{ER}}{\partial \mu} &= \frac{1 - \mu}{\left(2\mu^2 + 2\lambda^2 - 4\mu + 2\right)^{0.5}};\\ \frac{\partial \mu_{ER}}{\partial \lambda} &= \frac{-\lambda}{\left(2\mu^2 + 2\lambda^2 - 4\mu + 2\right)^{0.5}}\end{aligned} \tag{21}$$

On the other hand, if μ is less than λ, the resulting D_C is negative and, in this case, the derivatives are obtained as:

$$\begin{aligned}\mu_{ER} &= \frac{\left(2\mu^2 + 2\lambda^2 - 4\lambda + 2\right)^{0.5}}{2};\\ \frac{\partial \mu_{ER}}{\partial \mu} &= \frac{\mu}{\left(2\mu^2 + 2\lambda^2 - 4\lambda + 2\right)^{0.5}};\\ \frac{\partial \mu_{ER}}{\partial \lambda} &= \frac{\lambda - 1}{\left(2\mu^2 + 2\lambda^2 - 4\lambda + 2\right)^{0.5}}\end{aligned} \tag{22}$$

When μ is equal to λ, D_C results in zero. In this case, $D \geq 1$ and, as shown in Fig. 6, μ_{ER} is 0.5 and it is non-differentiable at this point. However, it is differentiable anywhere else and the partial derivatives at this point can be approximated by the mean of the derivatives from the right ($D_C > 0$) and the left ($D_C < 0$), such that:

$$\begin{aligned}\mu_{ER} &= 0.5;\\ \frac{\partial \mu_{ER}}{\partial \mu} &= \frac{1}{2\left(2\mu^2 + 2\lambda^2 - 4\mu + 2\right)^{0.5}};\end{aligned}$$

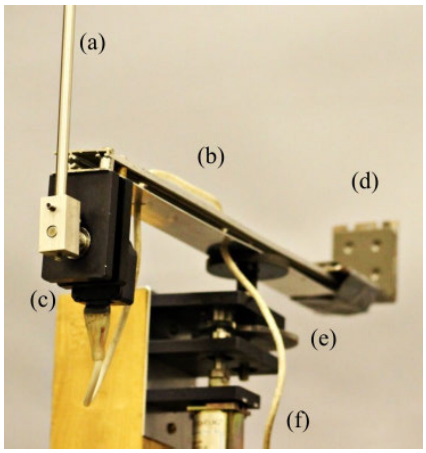


FIGURE 10. Prototype of RIP used for system identification. Pendulum (a), pendulum arm (b), pendulum angle encoder (c), counterweight (d), engine reduction (e), and DC motor with encoder for arm angle (f).

$$\frac{\partial \mu_{ER}}{\partial \lambda} = \frac{1}{2\left(2\mu^2 + 2\lambda^2 - 4\lambda + 2\right)^{0.5}} \tag{23}$$

When the "half" PAN, or linear output, is used, the derivatives are:

$$\begin{aligned} \frac{\partial \mu_{ER}}{\partial \mu} &= 0.5; \\ \frac{\partial \mu_{ER}}{\partial \lambda} &= -0.5 \end{aligned} \tag{24}$$

For example, based on Fig. 8a, the chain rule applied to the path from μ_{ER4} to μ_{1a} is given by:

$$\frac{\partial \mu_{ER4}}{\partial \mu_{1a}} = \frac{\partial \mu_{E1}}{\partial \mu_{1a}} \frac{\partial \mu_{E3}}{\partial \mu_3} \frac{\partial \mu_{ER4}}{\partial \mu_4} = 0.25 \frac{\partial \mu_{ER4}}{\partial \mu_4} \tag{25}$$

Multiple PAL2v neurons, as described here, could be connected to build a PNN according to the need. For the system (RIP) modeling, we designed one PNN with two PAL2v neurons of 4 inputs in the hidden layer and one PAN with linear μ_E in the output layer.

B. SYSTEM DESCRIPTION

We describe in this section the system to be identified by the proposed paraconsistent neural network.

The prototype has a control input u applied to the DC motor fixed to the arm. The encoders installed on the arm and pendulum provide the outputs related to the horizontal angle of the rotary arm (θ) and the vertical angle of the pendulum (α). The system is powered by a Teensy 3.2 microcontroller with a 32 bits Cortex-M4 processor, clocked at 72 MHz, flash memory of 256 kBytes, and RAM with 64 kBytes. Fig. 10 shows the RIP prototype used for this study, whose parameters are presented in Table 1.

This work does not intend to perform the mathematical description of the RIP, as well as the design methods for controlling it, which is well described elsewhere [26], [28], [36].

The prototype has only two measured outputs, which are the angles α and θ. Their derivatives at the k-th sample are

TABLE 1. Parameters of the RIP used for system identification.

SYMBOL	VALUE	DESCRIPTION
d	0.137 (m)	distance from the base axis to the center of mass
m_0	0.393 (kg)	mass of the arm
m_1	0.068 (kg)	mass of the pendulum
L_0	0.1825 (m)	length of half of the arm
L_1	0.1035 (m)	length of half of the pendulum
r	0.23 (m)	distance from base axis to the pendulum
R_m	2.4 (Ω)	motor armature resistance
K_t	0.08 (Nm/A)	torque constant of the motor, including reduction
K_e	0.2 (V/(rad/s))	velocity constant of the motor, including reduction
g	9.81 (m/s^2)	acceleration of gravity
θ	(rad)	angle of arm (positive for counter-clockwise
α	(rad)	angle of pendulum (positive for counter-clockwise
T_S	0.01 (s)	sampling time

approximated considering the Euler backward differentiation, such that,

$$\begin{aligned} \dot{\theta}_{[k]} &\approx \frac{\theta_{[k]} - \theta_{[k-1]}}{T_S}; \\ \dot{\alpha}_{[k]} &\approx \frac{\alpha_{[k]} - \alpha_{[k-1]}}{T_S}. \end{aligned} \tag{26}$$

Using the parameters of Table 1 and defining the state vector as $x = \left[\theta \ \alpha \ \dot{\theta} \ \dot{\alpha}\right]^T$, the resulting linear model is given by

$$\dot{x} = Ax + Bu, \tag{27}$$

with

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & -9.1047 & -0.6066 & 0.0132 \\ 0 & 86.2615 & 1.0110 & -0.1249 \end{bmatrix},$$

and

$$B = \begin{bmatrix} 0 \\ 0 \\ 31.6490 \\ -52.7483 \end{bmatrix}$$

A first-order low pass filter (LPF) is applied to all four outputs of the RIP, as presented in (28) for the arm angle. These filters reduce the noise of the RIP output signals. The sampling time (T_S) used is 0.01s, and we adopt the time constant (τ) as 0.05s.

$$\theta_{[k]filt} = \left(\frac{\tau}{\tau + T_S}\right)\theta_{[k-1]filt} + \left(\frac{T_S}{\tau + T_S}\right)\theta_{[k]} \tag{28}$$

After converting A and B matrices to discrete, we calculate the controller gain K by the pole placement method, considering the poles ate $[-2, -2.5, -11, -10]$, resulting in

$$K = \begin{bmatrix} -0.2160 & -5.3459 & -0.2563 & -0.5934 \end{bmatrix}.$$

The state feedback control law is given by $u_{[k]} = -K_{[k]}x_{[k]}$. The control signal ($u$) is the duty-cycle of the Pulse

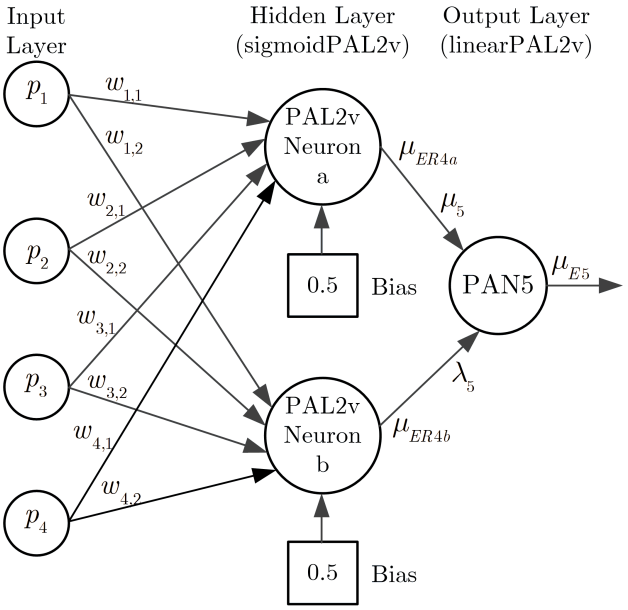


FIGURE 11. 4-Input PNN with 2 neurons in the hidden layer.

Width Modulation (PWM) considering also the direction of rotation, i.e., $u \in [-1;\ 1]$.

Since the RIP is highly unstable, we performed the identification in closed-loop control. We used three PNNs with four inputs and one output each, as presented in Fig. 11. Each PNN consists of two PAL2v neurons ($\text{summation}_{\text{PAL2v}}$) in the hidden layer, as shown in Fig. 8.a, combined with a PAN for the $\text{linear}_{\text{PAL2v}}$ at the output layer, providing one output.

We collected the input and outputs of the RIP to train the PNN in offline mode. Note that weights were applied only for the input layer of the PNN. Although this is not a requirement, we did not apply weights between the hidden to output layers.

First, we calculate the PNN output in the forward direction, taking the opportunity to calculate the neuron derivatives, as presented in Eqs. (21)-(24). We find the error between the target and PNN output. Next, we compute backward, applying the error correction to the PNN weights by gradient descent, using the chain rule example of Eq. (25).

The PAL2v is a comparison between two orthogonal inputs. Since the RIP in the closed-loop should be at equilibrium, the bias is fixed at 0.5, corresponding to $x = \begin{bmatrix}0 & 0 & 0 & 0\end{bmatrix}^T$.

IV. RESULTS

The following set of signals were used: control, pendulum and arm angles, pendulum and arm speeds $[u,\ \theta,\ \alpha,\ \dot{\theta},\ \dot{\alpha}]$.

Although we performed the learning process of the PNN offline, Fig. 12 presents the concept.

For the target output, the remaining signals are applied as inputs. Therefore, for the first paraconsistent neural net (PNN1) to learn the angle of the arm (θ) of the RIP, the inputs $[u,\ \alpha,\ \dot{\theta},\ \dot{\alpha}]$ are applied.

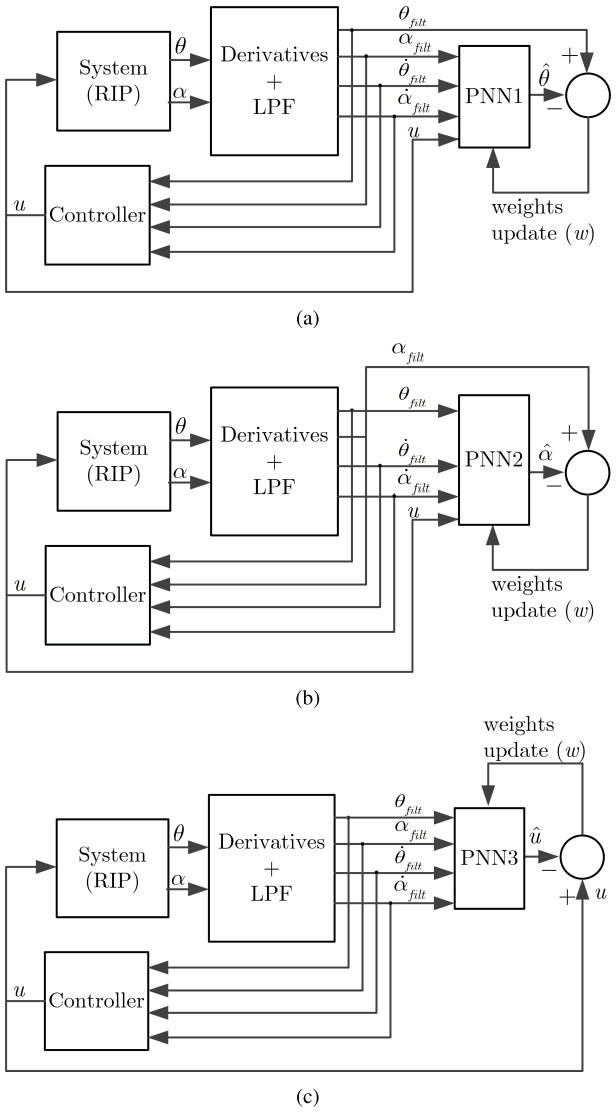


FIGURE 12. RIP and Controller identification with 3 PNNs. Arm angle model (a), pendulum angle model (b) and controller model (c).

The signals $[\theta,\ u,\ \dot{\theta},\ \dot{\alpha}]$ are used by a second paraconsistent neural net (PNN2) to model the angle of the pendulum (α).

Finally, the RIP outputs $[\theta,\ \alpha,\ \dot{\theta},\ \dot{\alpha}]$ are applied to the last paraconsistent neural net (PNN3) to model the control signal (u). So, PNN1 and PNN2 are used to model the RIP, also called Forward Model, while PNN3 results in the Inverse Model, or the controller model of the system, i.e, PNN3 is learning the control law.

During the experiment, we did not apply any reference for the arm to follow.

A. RIP AND WELL-TUNED CONTROL

We collected twenty thousand samples from the control signal (u) applied to RIP and its output $[\theta,\ \alpha,\ \dot{\theta},\ \dot{\alpha}]$. The mini-batch learning process is applied, using 2000 random values from the first 8000 samples to train the PNNs. We

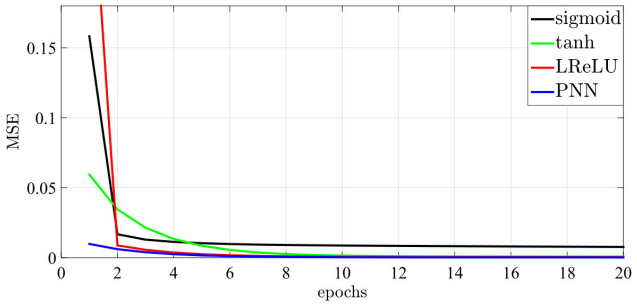


FIGURE 13. MSE of PNN and ANNs with different activation functions for the RIP arm angle modeling.

TABLE 2. MSE results for RIP ARM modelling.

ACTIVATION FUNCTION	MSE
sigmoid	0.0080377
tanh	0.0008452
LReLU	0.0003966
PNN	0.0001694

TABLE 3. MSE results for PNN and standard ANN.

NEURAL NET	MSE TRAINING	MSE VALIDATION	EPOCHS
ANN 2 neurons	3.99×10^{-5}	4.68×10^{-5}	105
ANN 4 neurons	8.45×10^{-5}	6.96×10^{-5}	119
PNN1	8.97×10^{-6}	7.18×10^{-5}	60
ANN 2 neurons	2.13×10^{-7}	3.29×10^{-7}	155
ANN 4 neurons	3.40×10^{-7}	5.14×10^{-7}	172
PNN2	2.07×10^{-7}	7.10×10^{-8}	10
ANN 2 neurons	3.69×10^{-6}	3.80×10^{-6}	233
ANN 4 neurons	2.82×10^{-6}	2.53×10^{-6}	277
PNN3	9.66×10^{-7}	4.33×10^{-6}	30

compared the PNN results with ANN in the same topology and the following activation functions: tanh, sigmoid, and LReLU. Fig. 13 presents the mean square error (MSE) results, based on the measurements to model the pendulum arm, as for PNN1.

A normalization step is required before we apply the signals to each ANN, and then we need to denormalize back at the output. The input signals were normalized within 0 to 1 for the PNN and sigmoid ANN and within -1 to 1 for tanh and LReLU, to compare the MSE.

We used the same input-output database, a learning factor of 0.25, and the same random starting weights for all neural nets. Fig. 13 suggests that the MSE results and the number of epochs required by PNN are comparable to the LReLU, followed by the tanh and sigmoid. Table 2 shows the MSE obtained after 20 epochs by each neural net in Fig. 13.

Although a speed comparison for the ANN algorithms has not been performed, sigmoid and tanh result in a simpler code, since they do not require an `if-then-else` structure in the routine, as required by LReLU and PNN.

In another comparison, three ANNs were created with the "neural net fitting" App (nftool) of Matlab, selecting 2 and 4 neurons in the hidden layer, and using the same database used for the PNNs. Table 3 presents the results.

The nftool creates a neural net using the tanh function in the hidden layer and linear activation function (purelin) in the output layer. The Scaled Conjugate Gradient method was selected for the learning process. Any other parameter was let also as default. The reason is that it is a trustworthy tool, with only a few parameters to be selected, such as the number of neurons, learning algorithm and % of points for training and validation. Besides both, ANN and PNN achieved good MSE results, Table 3 indicates that the paraconsistent neural net presented lower MSE results, requiring fewer epochs than standard ANN, even comparing ANN with 4 neurons in the hidden layer against PNN with only 2 neurons. The same is true for all the models: arm angle, pendulum angle, and controller output.

Fig. 14 presents the comparison between the RIP data and the models learned by the PNNs, for samples from 13000 to 14000, used for the validation process. One can see that the PNN could learn well the arm, pendulum, and controller outputs in this condition.

Fig. 14a shows the arm and pendulum speed. We can see the output of PNN1 compared to the arm of the RIP in Fig. 14b. Although we did not apply any reference signal, the figure shows that the arm angle is not static in 0 but presents an oscillation instead, relative to the controller design, as shown in Fig. 14b. This is mainly due to the derivative approximation and model uncertainties, such as actuator dead-zone, backlash, and Coulomb friction effects.

Fig. 14c presents the pendulum angle and the PNN2 output. The controller keeps the pendulum of RIP up, as the tiny variation of $+0.05$ to -0.075 rad.

Fig. 14d presents the output of the controller and the inverse model learned by PNN3. The output signals from the arm angle and the control have amplitudes of the same order of magnitude. However, Fig. 14d shows that PNN3 fits better the control signal, compared to Fig. 14b, for the arm angle signal. We see this result in Table 3, where the MSE for PNN1 and ANN with 2 and 4 neurons are in the order of 10^{-5}, while the control signal is in 10^{-6}. The results suggest that to model the arm angle, it is necessary to add more neurons.

B. LEARNED PNN AND DE-TUNED CONTROLLER

The PNN models, learned with a well-tuned controller, were compared to the system with a worse controller, with $K = \begin{bmatrix}-0.205 & -4.50 & -0.19 & -0.50\end{bmatrix}$.

Fig. 15 shows the results. We observe a performance degradation in this figure, compared to Fig. 14, due to the detuned controller.

Considering that the PNN1, PNN2, and PNN3 models learned with a well-tuned controller, Fig. 15b shows that PNN1 presents a behavior (blue) close to the pendulum arm (red). We observe a difference in amplitude, mainly in the negative half of the signal when the detuned controller is considered.

Fig. 15c shows that the PNN2 output (blue) is very close to the RIP pendulum angle output when we apply the detuned controller. The same is valid for Fig. 15d, where the inverse model output response (PNN3) (blue) follows the detuned controller output (red) closely. In this situation, the MSE for PNN1 is 0.0056, the MSE for PNN2 is 0.00006, and MSE for PNN3 is 0.0012. Based on these results, we can conclude

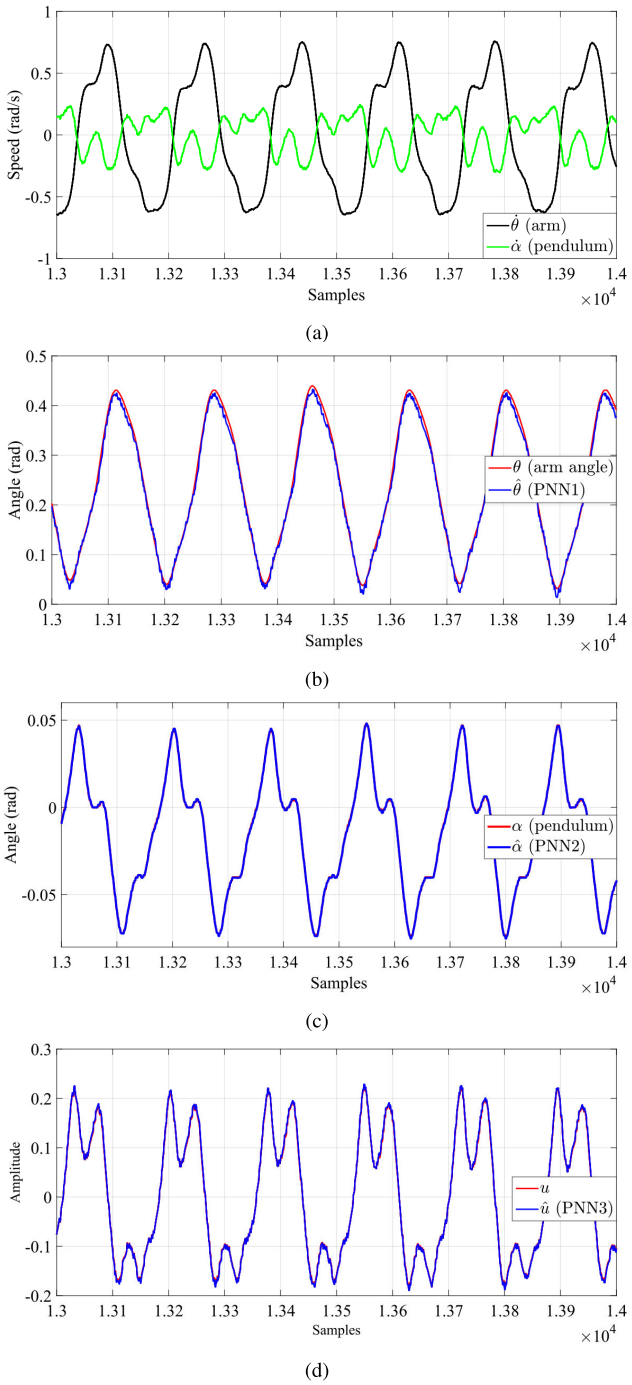


FIGURE 14. RIP with well-tuned controller: (a) Arm and pendulum calculated speed outputs, (b) Arm angle and PNN1 output, (c) Pendulum angle and PNN2 output, and (d) Controller signal, u, and PNN3 output.

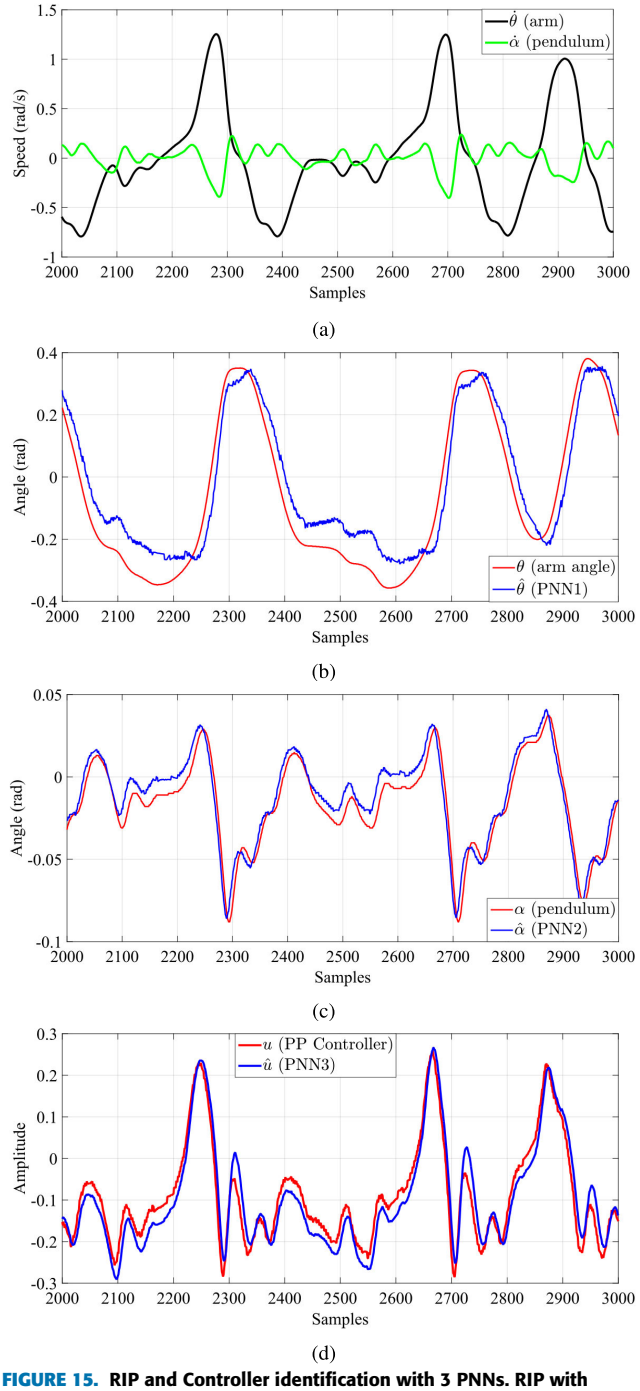


FIGURE 15. RIP and Controller identification with 3 PNNs. RIP with detuned controller: (a) Arm and pendulum speed outputs, (b) Arm angle and PNN1 model, (c) Pendulum angle and PNN2 model, and (d) Controller signal, u, and PNN3 model.

that the PNNs trained with a well-tuned controller could represent the system with a detuned controller, highlighting the generalization feature of the PNNs.

C. INVERSE MODEL CONTROL OF THE RIP BY PNN

Table 4 presents the weights learned by PNN1, PNN2, and PNN3 with the well-tuned controller, where a represents the first PAL2v neuron and b the second PAL2v in the hidden layer of each PNN, respectively.

We applied the input gain (G_I) of 0.25 and added a 0.5 DC level to all inputs to perform the normalization in the interval from 0 to 1. The outputs were also de-normalized by removing the DC level and applying the output gain (G_O) to all PNNs outputs, as presented in (29), where $N = 4$ is the number of Linear PAN used in the path of the PNN. In this

TABLE 4. Weights learned by PNN1, PNN2, and PNN3.

PNN	Neuron	w_1	w_2	w_3	w_4
PNN	a	4.0008	19.5192	0.8129	-1.0993
1	b	-3.9164	-19.2329	-0.8423	1.1936
PNN	a	0.0546	-0.2096	0.1631	0.0759
2	b	0.2082	0.2834	0.0599	0.2172
PNN	a	0.2020	-2.3940	0.1200	0.3052
3	b	-0.1085	2.4581	0.1074	-0.2291

study, we used three PNNs to create the PAL2v neuron of 4 inputs plus 1 linear PAN in the output layer.

$$G_O = \frac{1}{G_I} 2^N. \tag{29}$$

With the values of the weights learned by PNN3, we replaced the Pole placement controller with a simple PNN, which was embedded in the microcontroller of the RIP system. The goal here is to validate the PNN algorithm in replacing the existing controller, with the execution of the calculations of the PAL2v activation function on neurons at each sampling time. This is required due to the pendulum dynamics and the `if-then` statement required by the PAL2v neuron to calculate the μ_{ER} output at each sampling time, as presented in (15) to (17). Fig. 16 presents the results.

Fig. 16a shows that the PNN3 controller was able to control the RIP and keep the angle of the pendulum in a tiny variation (green line). The control applied by PNN3 to the RIP is in Fig. 16b.

Note that the amplitude and shape of the control signal ($\hat{u}$) presented in Fig. 16b is very close to the one shown in Fig. 16d using the well-tuned controller, indicating that the PNN learned correctly.

The robustness of PNN3 in control the RIP is verified by applying several disturbances in the pendulum, as seen at the sample 1400 of Fig. 16c. In Fig. 16d, we can observe the controller reaction and recovery of pendulum angle control. Note that the amplitude of signal $\hat{u}$ is limited within $[-1, +1]$.

Fig. 16e shows the power spectral density of the signals u (from pole placement control) and $\hat{u}$ (from PNN3), by fast Fourier transform (FFT) method, during 2000 samples.

The robustness of the PNN3 to control the RIP is a function of two main points: the quality of the controller used for the learning process and how low is the MSE obtained.

By increasing the number of neurons or the size of the database can result in lower MSE, with the compromise of increasing the time required by PNN for learning. On the other hand, a higher number of neurons can offer redundancy for the PNN.

The PNN3 controller ($\hat{u}$) algorithm implemented is:

- Compute PAL2v Neuron a

$$\mu_{4a} = G_I 0.25(w_{1,1}p_1 - w_{2,1}p_2 - w_{3,1}p_3 + w_{4,1}p_4) + 0.5$$

$$\mu_{4a} = \begin{cases} 0 & \mu_{4a} \leq 0 \\ 1 & \mu_{4a} \geq 1 \\ \mu_{4a} & \text{otherwise} \end{cases}$$

$$\lambda_{4a} = 0.5$$

$$d_{4a} = \left(\left(1 - |\mu_{4a} - \lambda_{4a}|\right)^2 + \left(\mu_{4a} + \lambda_{4a} - 1\right)^2\right)^{0.5}$$

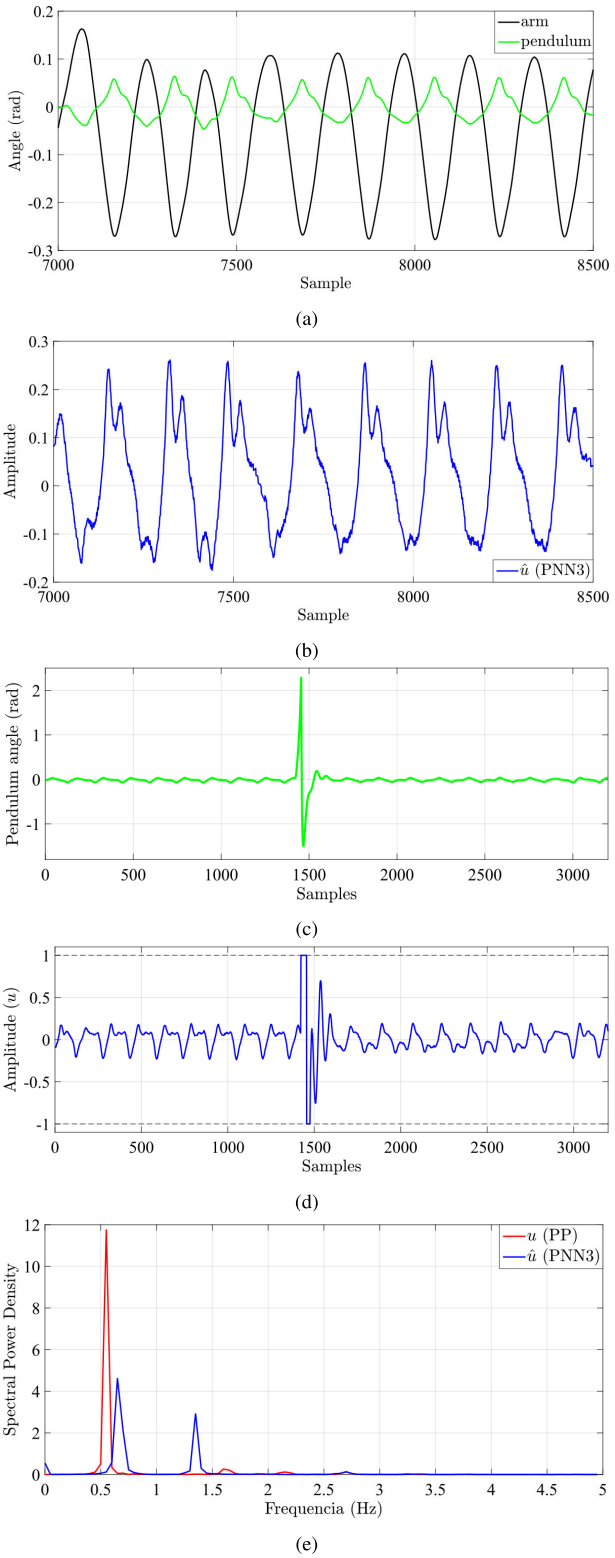


FIGURE 16. RIP is controlled by PNN3. Arm and pendulum angles (a), PNN3 controller output, $\hat{u}$ (b), pendulum under disturbance (c), PNN3 controller reaction (d), the spectral power density of the control effort (e).

$$D_{4a} = \begin{cases} 1 & d_{4a} \geq 1 \\ d_{4a} & \text{otherwise} \end{cases}$$

$$\mu_{ER4a} = \begin{cases} (2 - D_{4a})/2 & \mu_{4a} \geq \lambda_{4a} \\ D_{4a}/2 & \mu_{4a} \leq \lambda_{4a} \\ 0.5 & \text{otherwise} \end{cases}$$

- Compute PAL2v Neuron b

$$\mu_{4b} = G_I 0.25(w_{1,2}p_1 - w_{2,2}p_2 - w_{3,2}p_3 + w_{4,2}p_4) + 0.5$$

$$\mu_{4b} = \begin{cases} 0 & \mu_{4b} \leq 0 \\ 1 & \mu_{4b} \geq 1 \\ \mu_{4b} & \text{otherwise} \end{cases}$$

$$\lambda_{4b} = 0.5$$

$$d_{4b} = \left(\left(1 - |\mu_{4b} - \lambda_{4b}|\right)^2 + \left(\mu_{4b} + \lambda_{4b} - 1\right)^2\right)^{0.5}$$

$$D_{4b} = \begin{cases} 1 & d_{4b} \geq 1 \\ d_{4b} & \text{otherwise} \end{cases}$$

$$\mu_{ER4b} = \begin{cases} (2 - D_{4b})/2 & \mu_{4b} \geq \lambda_{4b} \\ D_{4b}/2 & \mu_{4b} \leq \lambda_{4b} \\ 0.5 & \text{otherwise} \end{cases}$$

- Compute PNN3 output ($\hat{u}$):

$$\mu_{ER5} = \frac{\mu_{ER4a} - \mu_{ER4b} + 1}{2}$$

$$\text{PNN3} = G_O\left(\mu_{ER5} - 0.5\right)$$

$$\text{PNN3} = G_O\left(\frac{\mu_{ER4a} - \mu_{ER4b}}{2}\right)$$

V. CONCLUSION

This investigation presents a methodology to build ANNs using the equations and rules of PAL2v, called here as PNN. The PAL2v activation function with two orthogonal inputs allows more flexibility for the designer to distribute the inputs. We trained the PNN by backpropagation with gradient descent. Although we can investigate other learning methods, this method proved to be good enough. The PNN required fewer epochs and achieved low MSE, compared to standard ANN using the hyperbolic tangent sigmoid function or Leaky-ReLU, in similar conditions. Although we built the neural networks with few neurons, based on the results, the PAL2v equations and rules can be used as activation functions, allowing the design of paraconsistent neural networks. The paraconsistent neuron algorithm requires some decision questions to calculate the D and μ_{ER} breaking point as some other functions, such as the ReLU based neurons. Although we did not perform a processing time assessment, the microcontroller runs the PNN algorithm, even with all if-then statements required by each PAL2v neuron, during a short sampling time (0.01 seconds) without losing control of the RIP. The results achieved here for the RIP suggest that PNN is an alternative for applications in the identification and adaptive control of nonlinear dynamic systems. With the mathematical framework presented in this study, the designers can propose more complex PNNs, either by increasing the number of paraconsistent cells and layers, configuring a recurrent paraconsistent neural network (RPNN), or building hybrid PNN + ANN for deep learning networks, allowing them to work in even more challenging engineering applications.

REFERENCES

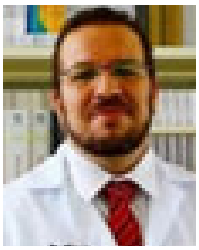
[1] M. T. Hagan, H. B. Demuth, M. H. Beale, and O. De Jesus, *Neural Network Design*, 2nd ed., M. Hagan, Ed. New York, NY, USA: Martin Hagan, 2014, p. 802.
[2] M. T. Hagan, H. B. Demuth, and O. D. Jesús, "An introduction to the use of neural networks in control systems," *Int. J. Robust Nonlinear Control*, vol. 12, no. 11, pp. 959–985, Sep. 2002.
[3] O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, N. A. Mohamed, and H. Arshad, "State-of-the-art in artificial neural network applications: A survey," *Heliyon*, vol. 4, no. 11, Nov. 2018, Art. no. e00938.
[4] Z. Fu, W. Xie, S. Rakheja, and J. Na, "Observer-based adaptive optimal control for unknown singularly perturbed nonlinear systems with input constraints," *IEEE/CAA J. Automatica Sinica*, vol. 4, no. 1, pp. 48–57, Jan. 2017.
[5] Y. Chen, J. Ren, and C. Yi, "Neural networks for the output tracking-control problem of nonlinear strict-feedback system," *IEEE Access*, vol. 5, pp. 26257–26266, 2017.
[6] A. Ebadat, P. E. Valenzuela, C. R. Rojas, and B. Wahlberg, "Model predictive control oriented experiment design for system identification: A graph theoretical approach," *J. Process Control*, vol. 52, pp. 75–84, Apr. 2017.
[7] M. Gevers, "Identification for control: From the early achievements to the revival of experiment design," *Eur. J. Control*, vol. 11, nos. 4–5, pp. 335–352, 2005.
[8] J. Schoukens and L. Ljung, "Nonlinear system identification: A user-oriented road map," *IEEE Control Syst. Mag.*, vol. 39, no. 6, pp. 28–99, Dec. 2019.
[9] M. S. Coelho, J. I. da Silva Filho, H. M. Côrtes, A. de Carvalho, M. F. Blos, M. C. Mario, and A. Rocco, "Hybrid PI controller constructed with paraconsistent annotated logic," *Control Eng. Pract.*, vol. 84, pp. 112–124, Mar. 2019.
[10] A. Y. Alanis, E. N. Sanchez, A. G. Loukianov, and M. A. Perez, "Real-time recurrent neural state estimation," *IEEE Trans. Neural Netw.*, vol. 22, no. 3, pp. 497–505, Mar. 2011.
[11] M. Stogiannos, A. Alexandridis, and H. Sarimveis, "Model predictive control for systems with fast dynamics using inverse neural models," *ISA Trans.*, vol. 72, pp. 161–177, Jan. 2018.
[12] N. da Costa and C. de Ronde, "The paraconsistent logic of quantum superpositions," *Found. Phys.*, vol. 43, no. 7, pp. 845–858, Jul. 2013.
[13] G. W. Favieiro and A. Balbinot, "Paraconsistent random forest: An alternative approach for dealing with uncertain data," *IEEE Access*, vol. 7, pp. 147914–147927, 2019.
[14] G. Priest, K. Tanaka, and Z. Weber, "Paraconsistent logic," in *The Stanford Encyclopedia of Philosophy*, E. N. Zalta, Ed. Stanford, CA, USA: Stanford University, Metaphysics Research Lab., 2018.
[15] J. I. D. S. Filho, "Treatment of uncertainties with algorithms of the paraconsistent annotated logic," *J. Intell. Learn. Syst. Appl.*, vol. 4, no. 2, pp. 144–153, 2012.
[16] J. M. Abe, K. Nakamatsu, and J. I. D. S. Filho, "Three decades of paraconsistent annotated logics: A review paper on some applications," *Procedia Comput. Sci.*, vol. 159, pp. 1175–1181, Jan. 2019.
[17] J. I. Da Silva Filho, G. Lambert-Torres, and J. M. Abe, "Uncertainty treatment using paraconsistent logic: Introducing paraconsistent artificial neural networks," in *Frontiers in Artificial Intelligence and Applications*. Amsterdam, The Netherlands: IOS Press, 2010, p. 311.
[18] D. V. Garcia, J. I. da Silva Filho, L. Silveira, M. T. T. Pacheco, J. M. Abe, A. Carvalho, M. F. Blos, C. A. G. Pasqualucci, and M. C. Mario, "Analysis of Raman spectroscopy data with algorithms based on paraconsistent logic for characterization of skin cancer lesions," *Vibrational Spectrosc.*, vol. 103, Jul. 2019, Art. no. 102929.
[19] Z. Ben Hazem, M. J. Fotuhi, and Z. Bingul, "A comparative study of the joint neuro-fuzzy friction models for a triple link rotary inverted pendulum," *IEEE Access*, vol. 8, pp. 49066–49078, 2020.
[20] X.-Y. Liu, R.-S. Jia, Q.-M. Liu, C.-Y. Zhao, and H.-M. Sun, "Coastline extraction method based on convolutional neural networks—A case study of Jiaozhou Bay in Qingdao, China," *IEEE Access*, vol. 7, pp. 180281–180291, 2019.
[21] A. Apicella, F. Donnarumma, F. Isgrò, and R. Prevete, "A survey on modern trainable activation functions," *Neural Netw.*, vol. 138, pp. 14–32, Jun. 2021.

[22] M. Everett, G. Habibi, and J. P. How, "Robustness analysis of neural networks via efficient partitioning with applications in control systems," *IEEE Control Syst. Lett.*, vol. 5, no. 6, pp. 2114–2119, Dec. 2021.
[23] S. Kizir, Z. Bingul, and C. Oysu, "Fuzzy control of a real time inverted pendulum system," *J. Intell. Fuzzy Syst.*, vol. 21, nos. 1–2, pp. 121–133, 2010.
[24] Z. B. Hazem, M. J. Fotuhi, and Z. Bingül, "Development of a fuzzy-LQR and fuzzy-LQG stability control for a double link rotary inverted pendulum," *J. Franklin Inst.*, vol. 357, no. 15, pp. 10529–10556, Oct. 2020.
[25] K. Furuta and M. Iwase, "Swing-up time analysis of pendulum," *Bull. Polish Acad. Sci., Tech. Sci.*, vol. 52, no. 3, pp. 153–163, 2004.
[26] C. Aguilar-Avelar and J. Moreno-Valenzuela, "A composite controller for trajectory tracking applied to the Furuta pendulum," *ISA Trans.*, vol. 57, pp. 286–294, Jul. 2015.
[27] N. Singh and S. Yadav, "Comparison of LQR and PD controller for stabilizing double inverted pendulum system," *Int. J. Eng.*, vol. 1, no. 12, pp. 69–74, 2012.
[28] P. Seman, M. Juh, and M. Salaj, "Swinging up the furuta pendulum and its stabilization via model predictive control," *J. Electr. Eng.*, vol. 64, no. 3, pp. 152–158, May 2013.
[29] D. Chandran, B. Krishna, V. I. George, and I. Thirunavukkarasu, "Model identification of rotary inverted pendulum using artificial neural networks," in *Proc. Int. Conf. Recent Develop. Control, Autom. Power Eng. (RDCAPE)*, Mar. 2015, pp. 146–150.
[30] P. Gautam, "System identification of nonlinear inverted pendulum using artificial neural network," in *Proc. Int. Conf. Recent Adv. Innov. Eng. (ICRAIE)*, Dec. 2016, pp. 1–5.
[31] R. B. R. Genal, "Nonlinear system identification and control: Rop scheme and neural networks," in *Proc. 2nd Int. Conf. Intell. Comput. Control Syst. (ICICCS)*, Jun. 2018, pp. 1831–1836.
[32] K. G. Vamvoudakis, F. L. Lewis, and S. S. Ge, "Neural networks in feedback control systems," in *Mechanical Engineers' Handbook*, M. Kutz, Ed. Hoboken, NJ, USA: Wiley, 2015.
[33] A. H. Jafari and M. T. Hagan, "Application of new training methods for neural model reference control," *Eng. Appl. Artif. Intell.*, vol. 74, pp. 312–321, Sep. 2018.
[34] A. Sharafian and R. Ghasemi, "Fractional neural observer design for a class of nonlinear fractional chaotic systems," *Neural Comput. Appl.*, vol. 31, no. 4, pp. 1201–1213, Apr. 2019.
[35] J. M. Abe and K. Nakamatsu, "Paraconsistent annotated evidential logic Eτ and applications in automation and robotics," in *The Handbook on Reasoning-Based Intelligent Systems*. Singapore: World Scientific, 2013, pp. 331–352.
[36] N. J. Mathew, K. K. Rao, and N. Sivakumaran, "Swing up and stabilization control of a rotary inverted pendulum," *IFAC Proc. Volumes*, vol. 46, no. 32, pp. 654–659, Dec. 2013.

ARNALDO DE CARVALHO, JR., is graduated in electrical engineering from Santa Cecilia University (UNISANTA), in 1991. He received the M.B.A. degree in business management from the Fundação Getulio Vargas (FGV), with executive management extension from the University of California at Irvine (UCI), Irvine, in 2001, and the master's degree in mechanical engineering from Santa Cecilia University (UNISANTA), in 2017. He is currently pursuing the Ph.D. degree in electrical engineering with the Escola Politécnica of the Universidade de São Paulo. He is also a Lecturer/a Researcher with the Federal Institute of Education, Science and Technology of São Paulo (IFSP), Cubatão, Brazil.

JOÃO FRANCISCO JUSTO received the B.Sc. and M.Sc. degrees in physics from the Universidade de São Paulo, in 1988 and 1991, respectively, and the Ph.D. degree in nuclear engineering from the Massachusetts Institute of Technology, in 1997. From 1997 to 1998, he was a Postdoctoral Fellow with the Massachusetts Institute of Technology and a Visiting Associate Professor with the University of Minnesota, from 2007 to 2008. He is currently a Full Professor of electrical engineering with the Escola Politécnica of the Universidade de São Paulo, Brazil. He has experience in materials science, focusing on computational modeling of nanomaterials and electrical engineering, focusing on embedded electronics and nanoelectronics. He was elected as a Full Member to the Alpha Nu Sigma Honor Society of the American Nuclear Society, in 1996, and the Sigma Xi—The Scientific Research Honor Society, in 1998.

BRUNO AUGUSTO ANGÉLICO received the B.Sc. degree from the State University of Londrina, in 2003, and the M.S. and Ph.D. degrees from the Universidade de São Paulo, in 2005 and 2010, respectively, all in electrical engineering. From 2009 to 2013, he was an Associate Professor with the Federal University of Technology of Paraná (UTFPR), Cornélio Procópio. He is currently an Assistant Professor with the Department of Telecommunications and Control Engineering, Escola Politécnica—Universidade de São Paulo. He has experience in electrical engineering, focusing on practical control applications and digital control.



ALEXANDRE MANIÇOBA DE OLIVEIRA (Member, IEEE) received the B.Sc. degree in electrical computing engineering from the Catholic University of Santos, in 2009, and the M.S. and Ph.D. degrees in electrical engineering from the Escola Politécnica of the Universidade de São Paulo, in 2012 and 2015, respectively. He is currently a Professor with the Department of Control and Automation Engineering, Federal Institute of Education, Science and Technology of São Paulo at Cubatão, where he is also the Coordinator of the James Clerk Maxwell of Microwave and Applied Electromagnetism Laboratory. He has experience in RFIC, microwave antennas, and scientific and technology education. He is a Full Member of the Council of the Brazilian Microwave and Optoelectronics Society (Management for the period 2019–2022).

JOÃO INÁCIO DA SILVA FILHO (Member, IEEE) received the bachelor's degree in electrical engineering with emphasis on electronics, the master's degree in electrical engineering, and the Ph.D. degree in electrical engineering with research in digital systems from the Escola Politécnica—Universidade de São Paulo, Brazil, in 1986, 1996, and 1999, respectively. In 2009, he did a postdoctorate research at the INESC Porto—Institute for Systems and Computer Engineering of Porto. He is currently a Professor with Santa Cecilia University, Santos, Brazil, and the Coordinator of the Group of Paraconsistent Logic Applied (GLPA).