A Processing Expansion Module For The LEGO MINDSTORMS NXT

Marcel Danilo Alves Siqueira[1], Helder Luiz Schmitt[2], Alessandro Goedtel[3] and Marcos Banheti Rabello Vallim[4]

***Abstract*— This paper presents a processing expansion module (MEP) for a NXT controller. The NXT computational capability is insufficient for some applications. In this work we introduce a LEGO module to support this performance demand. The module is connected to the NXT by a RS-485 communication. Through it, MEP sends requisitions to the NXT, which acts like a slave only. Into the MEP, NXC functions are instantiated in C language. With this new structure, the user's algorithm is processed by the MEP, then the it sends data with NXC functions parameters just for inherent resources of the NXT. The results show latency times of the instantiated functions, comparing them to another known remote processing methods. The MEP functionality is demonstrated in a practical application, where a line follower robot is controlled by an artificial neural network.**

I. INTRODUCTION

Educational robotics arose as a way of teaching of science and technology. Computational resources that prevailed in industrial and research environments were taken to education field [1]. Today, robotics is seen as a tool for teaching and catalysing study of technology [2]. Within the scenario of educational robotics there is an important character, the LEGO MINDSTORMS NXT. This product is the result of a series of evolutions, with work started by the fusion of LEGO mechanics with computational resources. With the NXT kit several applications are possible, for example, on teaching of control systems, nonlinear systems and embedded computing [3].

The LEGO mechanics friendliness and the availability of several and easy-to-use computing resources motivate the of use of NXT [4]. Commercially there are wide range of sensors and actuators developed and compatible with the NXT, which make it wanted for fast implementations that require advanced hardware: sensors of color , pressure, light, acceleration and electronic compass are examples of devices ready to use available for the NXT.

*This work was supported by the Federal University of Technology - Paraná (UTFPR), campus Cornélio Procópio.

[1,4] Marcel D. A. Siqueira and Marcos B. R. Vallim are with the Centro de Experimentação Ninho de Pardais, UTFPR, marceldanilo@gmail.com, mvallim@utfpr.edu.br.

[2,3] Helder L. Schmitt and Alessandro Goedtel are with the Centro Integrado de Pesquisa em Controle e Automação (CIPECA), UTFPR, 1640 Alberto Carazzai Avenue, Cornélio Procópio, Paraná, Brazil. helderschmitt@gmail.com, agoedtel@utfpr.edu.br.

The hardware design of the NXT kit dates from 2006. Even with the update version 2.0 there was no improvement in processing. This hardware profile can generate limitations for applications which require higher performance processing. To overcome this barrier, usually it is made by remote processing on a computer through Bluetooth communication [5], [4].

This paper presents an alternative solution to the performance limit of the NXT. The proposal is an onboard expansion module for remote processing, called here by MEP. Unlike other methods, the MEP uses RS-485 communication between the NXT and an external microcontroller, more modern and that has more powerful processing capability. In this structure, the NXT acts as a slave, restricted to the performance of functions required by the MEP. At the other end, an STM32F4 microcontroller concentrates most of the logic processing, and communicates with LEGO only to perform inherent NXT functions. To the user, the remote interface employs the syntax of the language NXC, already widely used with MINDSTORMS controllers.

Forward, we present details about the NXT kit hardware and communication. The design of the MEP is therefore presented, focusing on the communication strategy, the underlying hardware and software steps. Finally, the achieved results are shown, divided into an analysis of time response of remote functions and a practical application, with a line follower robot controlled by an embedded artificial neural network.

II. The LEGO MINDSTORMS NXT

The LEGO MINDSTORMS NXT kit has a set of mechanical parts and a set of hardware that contains the controller, sensors, actuators and wires. The combination of these components allows the creation of several types of robots, such as robotic arms, vehicles and inverted pendulum. The MINDSTORMS series, originally designed for child audiences, achieved wide audience of engineers and researchers [10].

A. Hardware NXT

The NXT controller hardware has two microcontrollers, the main one and the auxiliary one. The main microcontroller is an AT91SAM7S256, whose role is user's algorithm processing and communication. The auxiliary one, an ATMEGA48, is intended for low-level control of actuators and reading analog sensors. The

general structure of the NXT controller is illustrated in Figure 1.

The main microcontroller has a 32-bit ARM7 processor, operates at 48 MHz, has 256 kB of flash memory for programs and 64 kB RAM memory. The device is responsible for storage and execution of firmware, which implements functions to access the hardware and runs user applications.

B. NXT communication resources

To exchange data between the NXT and any other computer, the following means are commonly used: I2C, RS-485, USB and Bluetooth - all interfaced with the main microcontroller.

The NXT brick has four sensor inputs, and any of these can implement I2C communication. In brick, the maximum transmission rate for the I2C is 9600 bps, with packages of eight bits [6].

The LEGO NXT has a Bluetooth Bluecore^{TM} 4.0 module, with up to 464 kbps [7]. For remote processing, Bluetooth has unstable performance, being vulnerable as the position of the brick (on a mobile robot) varies to the remote station location.

The main microcontroller of the NXT is a peripheral USB Full Speed, assuming maximum nominal 12 Mbps. Currently, the NXT has USB drivers for Windows, Mac and Linux operating systems only, restricting its use for these platforms.

Specially at the port number 4, the NXT brick implements RS-485 communication. This port reaches up to 921.6 kbps and can have a structure of a master and up to seven slaves [8]. Among all others, it is chosen for use in this work.

III. MEP CONCEPTION

The MEP is intended to support applications of bigger computational demand that the NXT can do. The applications in question are those that require more data processing than NXT resources. In applications such as these, the NXT processor can represent an obstacle for the performance of the robot. If the data processing is improved, there are a reduction of this obstacle and an increasing of the system performance. With this intent the expansion module is designed. This section organizes the structure of the solution, showing the necessary hardware and software for the embedded remote processing.

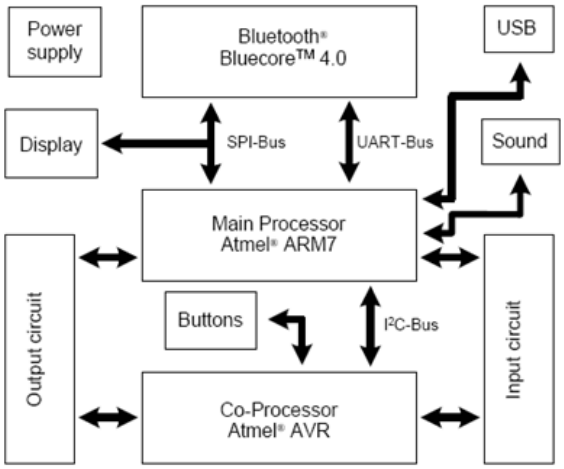

Fig. 1. NXT hardware structure. Extracted of [12]

A. Hardware Structure

The design of the module is based on a structure that has three main elements: the NXT, a communication and an external microcontroller. The hardware developed basically focuses on the RS-485 communication, the microcontroller powering, while the brick is not modified.

1) RS-485 Communication: The RS-485 communication is implemented in NXT using a converter ST485. In this component, TTL level voltages are converted to levels agreed for the RS-485.

The elements connected to the network do not need to share the negative reference, since only the differential voltage is viewed by a network element. It gives robustness to noise and stability for long cables communications [9].

In this work, we adopt the structure point-to-point network, among the some possible topologies for RS-485. The line is composed by the wires A and B. Once connected to the line AB, an element initiates a byte reception when the voltage $V_B - V_A$ is at least 0.2 V. Since there are only two elements in the network, there is no need for addressing.

Compared with other forms of communication on the NXT, the RS-485 stands out for ease of implementation and performance. Software support for RS-485 is sufficient for the development of the work, with functions for reading and writing bytes and communication control flags.

2) STM32F4 Series: To compose the MEP, some microcontrollers were evaluated. Among them, the STM32F407VG stood out due to its high-performance, low power consumption and a development board which can be embedded. It's employed a Cortex-M4F, working at 168 MHz clock, and performs up to 210 DMIPS (Dhrystone Million Instructions Per Second). The Cortex-M4F contains DSP (Digital Signal Processor) and an FPU (Float Point Unit) implemented in hardware.

To implement the RS-485 with STM32F4 was used UART4 module with its TX and RX pins, and pin GPIO (General Purpose Input Output) to data flow control. Additional pins were also used to interface buttons and LEDs. The work with this microcontroller was based on the STM32F4DISCOVERY development board, and it was developed an additional shield to implement the RS-485 functionality, so with a MAX485 transceiver.

B. Software Structure

The usability of the kit is one of the greatest benefits mentioned by users. There are structured programming

languages developed with specific features for the NXT. Thus, as we gained in processing, we aimed to keep the software usability. Among some several available, the NXC (Not Exactly C) was chosen, a free to use, widely spreaded language similar to C.

1) Instantiated Functions: The NXC language functions have syntax like this: $OnFwd(OUT\_AB,\ 50)$, a function which activates the outputs A and B at 50% of the rated speed. Internally, this function is passed to the auxiliary microcontroller (ATMEGA48), that performing the required control reduces the computational load of the NXT main processor.

The software strategy adopted in this work is similar. In a data packet bytes are sent; their values are function parameters, as exemplified by the equation (1). In this equation, the byte par_0 receives a constant set to a function, and the remaining bytes carry parameters of the corresponding function. That is the way the MEP makes its requests to NXT. This, in turn, responds with one/two-bytes packet only, for communication ACK (acknowledgement) or data return.

$$\{par_0,\ par_1,\ par_2 \ldots\ , par_6\} = \{\_OnFwd,\ OUT\_BC,\ 95,\ 0,\ 0,\ 0,\ 0\} \quad (1)$$

Since a function receives an identification number, the quantity functions instantiated in MEP is limited. So, in an eight-bit communication, up to 256 functions may be instantiated in the form of equation (1). This interval was divided into function groups, and each group received some of the most frequent used and most comprehensive functions of the NXC language.

2) Master-Slave Communication: The user program is written in C language in a programming environment for ARM microcontrollers. Instantiated functions are defined in a header file, that is common to the STM32F4 and NXT code compilers, IAR EWARM 6.3 and Bricx Command Center 3.3, respectively.

The software design of the MEP contains separate modules for (a) NXC functions, (b) user application, (c) communication and (d) startup. In (a) functions NXC are instantiated; once these functions are called from (b), the module output buffers UART4 is update with function parameters ($par_{0:6}$). From (c) a communication function is finally called, so it sends the parameters through RS-485 communication.

On the side of the NXT, when the communication bus is idle, an infinite loop waits for data to fulfil the 7-byte input buffer. When a packet is received, the first byte (par_0) is tested, and the program is directed to perform the corresponding function, carrying the other parameters received ($par_{1:6}$).

IV. PERFORMANCE AND APPLICATION ANALYSIS

The remote processing of NXT, embedded or remote, has performance related to the communication efficiency. If the gain in processing time is lost in communication delay, there is no real gain. This section aims to demonstrate and situate the gain we get on the MEP. The results cover aspects about communication time and an embedded neural network application.

A. Response Times Results

The speed of the RS-485 communication has been parameterized at 921.6 kbps, but the time response of a single function is a more important indicator to check the performance of the MEP. That time varies according to the requested functions, as the NXT, internally, makes different communications between microcontrollers, sensors, and actuators. Thus, the remote control performance via any communication type is dependent to the internal latencies of the NXT. The following functions are shown for motor control and sensor reading, and they present also a brief about the NXT internal communication:

- $OnFwd(OUT\_A, 100)$: sends a setpoint of 100% speed to the output A. Internally, these parameters are sent to ATmel48 via I2C;
- $Sensor(IN\_1)$: returns the value of a sensor set at the input 1. In this experiment, we set a light sensor whose reading is taken by ATmel48 and transferred to AT91SAM7 via I2C;
- $SensorUS(IN\_1)$: returns the value of the ultrasonic distance sensor configured to input 1. The sensor communicates directly with AT91SAM7 via I2C at the rate of 9.6 kbps.

To compare the performance of the proposed expansion module to standard tools, the three functions mentioned were tested. The Matlab toolbox RWTH-NXT was used to call functions from a personal computer, so its remote processing is linked to the NXT via either Bluetooth or USB. From the MEP, the same functions are called through the RS-485 port. The results are compared in Table I.

TABLE I
AVERAGE TIMES OF FUNCTIONS CALLED REMOTELY

Mode	OnFwd	Sensor	SensorUS
Bluetooth	17,7 ms	72,5 ms	138 ms
USB	16,2 ms	2,58 ms	19,8 ms
RS-485	3,26 ms	83,2 μs	19,0 ms

From Table I shortest average for RS-485 can be observed, as it carries packets with only seven bytes (Eq. 1) in contrast to Bluetooth and USB communications, which using packets 62 and 64 bytes, respectively. In terms of average values, the times of the RS-485 were lower even than those presented by USB, whose specification is much higher.

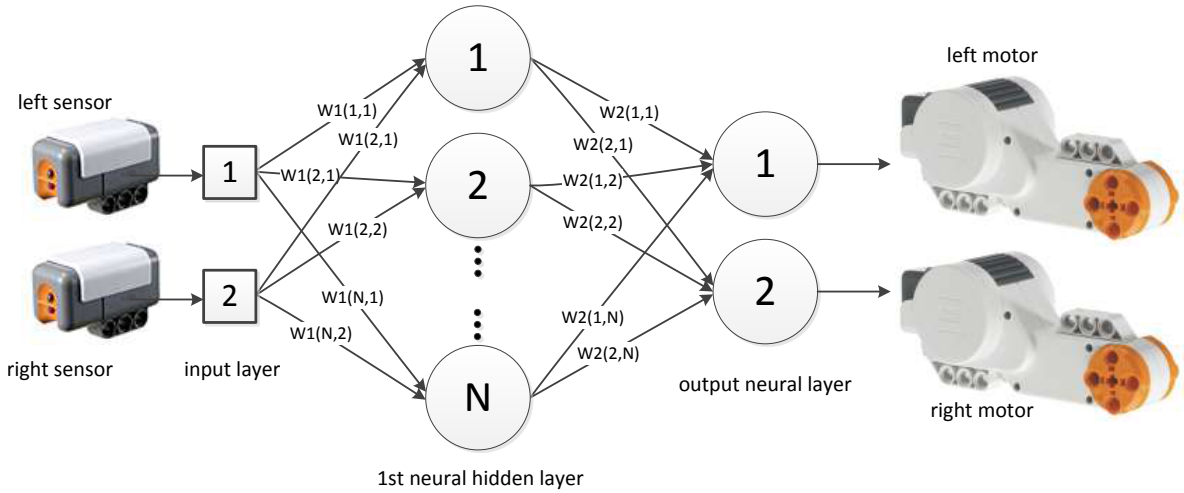

Fig. 2. MLP as controller of the robot

B. Application: NXT vs. MEP vs. PC

This subsection provides a comparative analysis of performance of an application running in three different computational means: LEGO NXT, MEP and PC. The goal is to show the behavior and performance of the hardware module proposed in this work, comparing it to LEGO NXT and the PC.

The application was based on a work of [5], where Artificial Neural Networks (ANN) were processed on the PC and communicated by the RWTH-NXT toolbox to the brick via Bluetooth. That study compared different types of networks such as control agents of a line follower robot. In this work, we employed a basic topology case of a Multi Layer Perceptron network (MLP). Once it has no delayed inputs, results may more sensitive reflect the processing performance for the task of line following.

1) MLP network: The composition of the MLP and the elements used in and out of the network are shown in Figure 2. In this network, two sensors values are read and normalized by feeding the first layer. The first hidden neural layer has a number of N neurons. The output neural layer contains two neurons, whose values go through a post-processing task and update the speed values of the two motors.

The robot was trained on a track, whose dimensions are illustrated in Figure 3. The excerpts from A to F are used for analysis of robot navigation. On this track, data were generated from a PID control for the network training. The values generated consisted of the error between two light sensors and output, with the velocity values of the two engines.

With this database, we used the training algorithm of the generalized delta rule, presented by [11]. We used learning rate $\eta = 0.1$, and error tolerance $\epsilon = 10^{-7}$. For every neuron, a logistic function was used, with $\beta = 1$.

2) Computational Load: The use of Artificial Neural Networks (ANN) in this paper serves two ideas: it is a typical application as a robot controller, and yet it

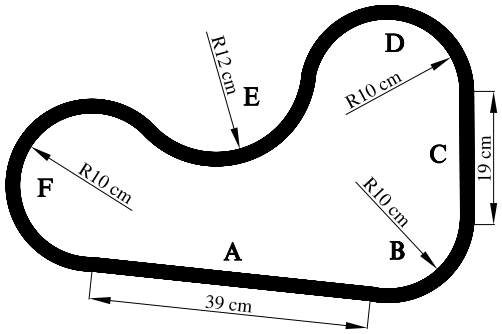

Fig. 3. Track to the navigation analysis

can represent a computational load. ANN in the Figure 2 has N neurons in the first hidden neural layer and this parameter is set to five, 25 and 50 neurons. The effect of this variation is the amount of operations to generate the outputs from the first hidden layer neural, as the equation 2 links. As result, there is a correspondent increasing of operations to the output neural layer, which has similar equation. Still about computational load, we aimed to evaluate the FPU effects over the variation of neurons number. In the equation, $g()$ is the logistic function. To implement embedded (NXT and MEP), this function was approximated by numeric table.

$$Y^{(1)} = g\left(\sum_{i=0}^{N} W_{ij}^{(1)} x_i\right) \tag{2}$$

3) Results for Iteration Times: The processing of the MLP was measured when performed in the three different computational means. The numeric indicator chosen is the value of the MLP iteration. When processing the NXT, there are no external communication times, just for the MEP implementations and PC, so it was possible to measure the MLP iteration with no communication. We called it here by insulated operation.

The results are shown in Table II. For the MLP in

TABLE II

ITERATION TIMES RESULTS OF THE MLP

Processor	operation: Normal	Insulated	FPU	Optimization	N = 5	N = 25	N = 50
NXT	✓			✓	20,2 *ms*	63,9 *ms*	127 *ms*
PC		✓	✓	✓	316,9 μs	319,9 μs	323,5 μs
	✓		✓	✓	202,6 *ms*	204,5 *ms*	206,0 *ms*
MEP		✓			31,4 μs	117 μs	225 μs
		✓		✓	22,7 μs	88,4 μs	170 μs
		✓	✓		11,1 μs	42,7 μs	82,2 μs
		✓	✓	✓	4,58 μs	17,0 μs	34,5 μs
	✓		✓	✓	17,00 *ms*	17,11 *ms*	17,22 *ms*

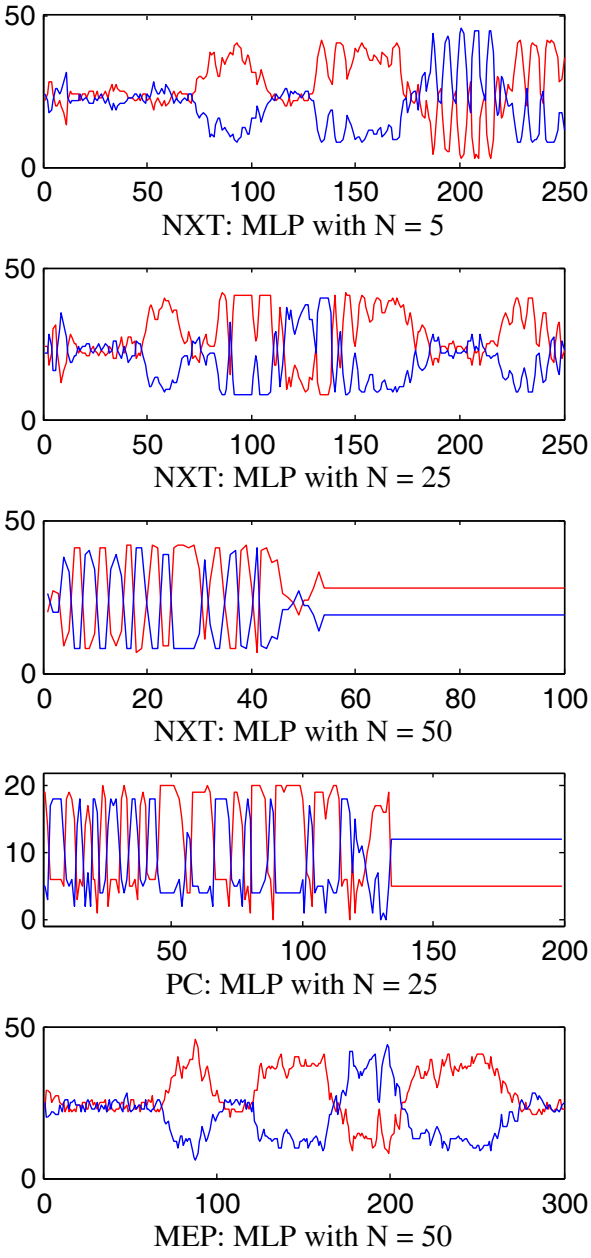

Fig. 4. Navigation response. In red color, the right motor speed; in blue, the left motor speed

the NXT, time was measured over 1000 iterations by an internal counter. At PC, it was used the functions tic/toc of the Matlab. For the MEP, time was measured by an oscilloscope Tektronix TDS-5054 over an output pin, which received an inversion of the network every 1000 iterations.

Comparing the figures, it is possible to observe the sensitivity of the NXT when the computational load gets increased, while the PC time values do not suffer significant changes, and the MEP remain virtually unchanged. With this comparison, there was observed advantages of the use of FPU and compiler code optimization.

4) Robot Navigation Results: The robot navigation was analyzed in NXT, PC and MEP. The results of the navigation performance were different, as suggested by the results in Table II. Graphical results are shown in Figure 4 for different implementations. The signals are shown as motors speed A and B. The purpose was to demonstrate three things: the failure of the NXT by increasing the computational load, the MEP performance, and the communication inefficiency with the PC.

The three first graphs show the behavior of the NXT. The first graph shows satisfactory behavior when iterating the network is 20.2ms (Table II) as the robot presents stable navigation. With the MLP of $N = 25$, the robot begins to unsettle the trajectory. For $N = 50$, the trajectory is lost at the beginning of section D of the circuit.

The implementation on the PC was not successful. The speed was limited to 60% in the output of the MLP for the robot did not leave the path. Thus, the robot intermittently suffered errors of navigation. Even when in straight movement, the robot was quite oscillatory (section A), which confirms the impact of long-time iteration for this implementation.

For the MEP, the iteration times did not vary significantly, so navigation was similar for all values of N assayed. The last graph shows the most critical case with $N = 50$. The result was able to replicate the behavior of the training.

V. CONCLUSIONS

This paper presented an expansion module for the LEGO NXT. The module aims to contribute to the ap-

plicability of the kit and for supporting applications that do not find sufficient capacity on the original hardware of the NXT. The results showed benefits when embedding remote processing using RS-485.

The design of the module occurred in a research environment for educational robotics, so some future works should aim to control applications of mobile robots for different topologies of ANN and other intelligent systems.

An important next step in the work will be migrating to free software for programming STM32F4 of the MEP. Currently, there is some tools emerging, like environments aimed to Arduino platforms, but that can be integrated to the STM32F4 programming.

ACKNOWLEDGMENT

Friends and workmates involved, thanks for all discussions and contributions.

References

[1] Foundation Logo, What is Logo?, 2011, available at `el.media.mit.edu/logo-foundation/logo/index.html`.

[2] M. B. R. Vallim, J. M. Farinesand and J. E. R. Cury Practicing engineering in a freshman introductory course, IEEE Transactions on Education, vol. 49, pp. 74-79, Feb. 2006.

[3] B. S. Heck, N. S. Clements and A. A. Ferri, A LEGO experiment for embedded control system design, IEEE Control Systems, vol. 24, pp. 61-64, Oct. 2004.

[4] C. J. Pretorius, M. C. du Plessis and C. B. Cilliers, A Neural Network-based kinematic and light-perception simulator for simple robotic evolution, Proceedings of the 2010 IEEE Congress on Evolutionary Computation(CEC), Jul. 2010.

[5] H. V. D. Silva, W. S. Gongora, A. Goedtel and M. B. R. VALLIM, Um estudo comparativo entre arquiteturas neurais aplicadas a um robô autônomo em trajetória orientada, Anais do XIX Congresso Brasileiro de Automática, Sep. 2012.

[6] M. Gasperi and P. Hurbain, Extreme NXT: Extending the Lego Mindstorms NXT to the Next Level, 2nd ed., Apress, New York City, 2009.

[7] S. Toledo, Analysis of the NXT Bluetooth-Communication Protocol, 2006, available at http://www.tau.ac.il/ stoledo/lego/btperformance.html.

[8] A. Shaw, Class RS-485, 2012, available at `www.lejos.sourceforge.net/nxt/nxj/api/lejos/nxt/comm/RS485.html`.

[9] J. Axelson, Networks for Monitoring and Control Using an RS-485 interface, Microcomputer Journal, vol. 1, pp. 27, 1995.

[10] P. Wallich, Mindstorms: not just a kid's toy, IEEE Spectrum, vol. 38, pp. 52-57, Sep. 2001.

[11] I. N. Silva, D. H. Spatti and R. A. Flauzino, Redes Neurais Artificiais para Engenharia e Ciências Aplicadas - Curso Prático, 1st ed., Artliber, 2010.

[12] E. Obrien, Teaching Critical Engineering and Robotics Concepts Using LEGO MINDSTORMS NXT and LabVIEW, 2012, available at `sine.ni.com/cs/app/doc/p/id/cs-12501`.