Autonomous Robots (2023) 47:947–961
https://doi.org/10.1007/s10514-023-10111-x

Towards neuromorphic FPGA-based infrastructures for a robotic arm

Salvador Canas-Moreno[1] · Enrique Piñero-Fuentes[1] · Antonio Rios-Navarro[1] · Daniel Cascado-Caballero[1] · Fernando Perez-Peña[2] · Alejandro Linares-Barranco[1]

Received: 30 July 2021 / Accepted: 22 May 2023 / Published online: 14 July 2023

Abstract

Muscles are stretched with bursts of spikes that come from motor neurons connected to the cerebellum through the spinal cord. Then, alpha motor neurons directly innervate the muscles to complete the motor command coming from upper biological structures. Nevertheless, classical robotic systems usually require complex computational capabilities and relative high-power consumption to process their control algorithm, which requires information from the robot's proprioceptive sensors. The way in which the information is encoded and transmitted is an important difference between biological systems and robotic machines. Neuromorphic engineering mimics these behaviors found in biology into engineering solutions to produce more efficient systems and for a better understanding of neural systems. This paper presents the application of a Spike-based Proportional-Integral-Derivative controller to a 6-DoF Scorbot ER-VII robotic arm, feeding the motors with Pulse-Frequency-Modulation instead of Pulse-Width-Modulation, mimicking the way in which motor neurons act over muscles. The presented frameworks allow the robot to be commanded and monitored locally or remotely from both a Python software running on a computer or from a spike-based neuromorphic hardware. Multi-FPGA and single-PSoC solutions are compared. These frameworks are intended for experimental use of the neuromorphic community as a testbed platform and for dataset recording for machine learning purposes.

Keywords Neuromorphic engineering · Spike-based motor control · FPGA · Robotic arm

Mathematics Subject Classification 68 · 92 · 93 · 94

1 Introduction

The effectiveness and efficiency of the central and peripheral nervous systems is the inspiration of Neuromorphic Engineering (NE). In this field, engineers are inspired by biology to solve engineering problems through mimicry. The hardware implementation of neuromorphic systems often requires fully customized solutions to develop the desired application. Typical technologies are Application-Specific Integrated Circuits (ASICs) (Chicca et al., 2014), Field Programmable Gate Arrays (FPGAs) (Maguire et al., 2007), or even Field Programmable Analog Arrays (FPAAs) (Rocke et al., 2008). Neuromorphic hardware developed in the last decade can be classified mainly into sensors and neural networks. In the field of sensors, the most representative examples are vision sensors (Lichtsteiner et al., 2008; Serrano-Gotarredona and Linares-Barranco, 2013; Gallego et al., 2020) audition sensors (Chan et al., 2007) and olfactory sensors (Koickal et al., 2007). The main analog neural processors are: Reconfigurable On Line Learning Spiking (ROLLS) Neuromorphic Processor (Qiao et al., 2015), Neurogrid (Benjamin et al., 2014), High Input Count Analog Neural Network chips (HICANNs) (Schemmel et al., 2008) (Schemmel et al., 2010) (Calimera A., , 2013) and Dynamic Neuromorphic Asynchronous Processor (Dynap-SE[1]) (Moradi et al., 2018). Digital implementations include the Spiking Neural Network Architecture (SpiNNaker) (Furber et al., 2014), the Loihi digital spiking processors from Intel (Davies et al., 2018) and

✉ Salvador Canas-Moreno
scanas@us.es

1 Robotics and Tech. of Computers Lab, Smart Computer Systems Research and Engineering Lab (SCORE), ETSII-EPS, Research Institute of Computer Engineering (I3US), Universidad de Sevilla, 41012 Seville, Spain

2 School of Engineering, Universidad de Cádiz, 11519 Puerto Real, Cadiz, Spain

1 https://www.ini.uzh.ch/en/research/groups/ncs/chips/Dynap-se.html (accessed on 26/Feb/2021)

Table 1 Comparison of hardware technologies for the neuromorphic field. FPAA power consumption citation (Selow et al., 2009)

	ASIC	FPGA	FPAA
Reconfiguration	Limited	Full	Possible
Domain	Mixed	Digital	Analog
Neuron density	The biggest	Big	Small
Power consumption	Smallest	Small	Small
Performance	The best	Good	Moderate

the TrueNorth chip from IBM (Cassidy et al., 2013) as the main representatives.

Table 1 sumarizes the pros and cons of these three technologies typically used in this field. ASICs are not reconfigurable, thus the manufactured circuit is permanent, and the price of an ASIC is typically high. On the other hand, an ASIC has several advantages; for instance, they enable denser implementations (i.e., the largest population of neurons per chip), they provide better performance and power consumption than other hardware, and, since the logic is already implemented, they only require working on the software. Regarding the second technology, i.e., Field Programmable Gate Arrays (FPGAs), according to (Harkin et al., 2008) and (Maguire et al., 2007), the main drawback of the FPGA is that existing FPGAs can only provide limited synapse density. In contrast, FPGAs present an excellent prototyping alternative providing a high degree of parallelism in application execution; they enable many input/output connectors, as well as configurable and dedicated computational Spiking Neural Network (SNN) blocks which offer a suitable reconfigurable platform for SNN implementation, with optimal use of hardware area and power with a lower cost. Lastly, Field Programmable Analog Arrays (FPAAs), as stated in (Rocke et al., 2008), have several drawbacks: a limited amount of on-chip real estate restricts neuron density, a large number of devices are required to create a large scale reconfigurable analog SNN platform, and they enable few input/output connectors. However, in general, multiple development boards can be easily daisy-chained together to increase the reconfigurable analog area without the need to design custom VLSI circuitry.

Arrays of spiking neurons and complex neuromorphic architectures must be properly and efficiently integrated, thus a strategy for communicating spikes was needed in this field. Address-Event-Representation (AER) was proposed as a communication protocol for communicating spikes on a multiplexed digital bus across neural arrays (Sivilotti, 1992), which is presently a standard in the neuromorphic community. Address-Events (AEs) are digital events communicated with a four-step asynchronous protocol and a digital label (i.e., the address of the neuron in the array). The time between events is self-represented and the addresses identify the source neuron of the event. The use of mapping tables and switches / routers (Zamarreno-Ramos et al., 2013) allow for the routing of events to different destinations, enabling the design of complex and arbitrary topologies of SNNs.

Today's robotics are typically built with Commercial Off-The-Shelf (COTS) components that do not provide specific solutions for neuromorphic systems. Available products in the market provide motor controllers as black boxes, which receive a reference command for targeting a specific position or revolution speed of a joint. These controllers communicate with each other through industrial field buses, such as Controller-Area-Network (CAN), which introduces additional latency in the control loop and forces a fixed and not-so-optimal power consumption (Dominguez-Morales et al., 2011). Nevertheless, these robots are also used in the neuromorphic community as the last step towards their demonstrations of spike-based motor controllers or SNNs trained for robot control, like (Stagsted et al., 2020a, 2020b; Stroobants et al., 2021) using Loihi, (Zaidel et al., 2021; DeWolf et al., 2016) using the neuromorphic engineering framework (NEF), (Denk et al., 2013; Galluppi et al., 2014) using SpiNNaker or (Milde et al., 2017; Blum et al., 2017) using ROLLS spiking processor, among others, where the final actuation to the robot is through the vendor interface, which lacks the similarity with neural actuation at the muscle level.

To include the last step of using spikes to drive motors and produce complete sensory-processing-actuating neuromorphic spiking systems, direct access to the signals that drive the motors of the robots is required. Typically, these COTS motor controllers use Pulse-Width Modulation (PWM) to drive the motors, imposing a constant power consumption even when the joint is not moving, and an additional latency, due to the period of the PWM signal. In contrast, bio-inspired motor control spiking systems (Perez-Pe?a et al., 2013) aim to reduce this power consumption by reducing the activity of the signal driving the motors through the application of a Pulse Frequency Modulation (PFM) bio-inspired technique (Jimenez-Fernandez et al., 2012). This technique prevents the additional latency caused by the period of PWM.

In this work, we used an improved version of the neuromorphic robotic arm (from spiking sensors to spiking actuation) platform, called ED-Scorbot (Gómez-Rodríguez et al., 2016). This platform was initially composed of a set of Spartan FPGA-based boards for neuromorphic processing (developed at RTC lab in Seville) and a modified Scorbot ER-VII. The original controller of the robot was replaced by a couple of neuromorphic FPGA-based boards for implementing spiking motor controllers and allow commanding the robot from both a neuromorphic spiking hardware system and a dedicated computer. The framework was constructed to provide full-access to both the sensors and the actuators in a spike-based manner. For each joint of this robot, an improved version of the spike-based PID controller integrated in Gómez-Rodríguez et al. (2016) and Linares-Barranco et al. (2020) was applied, and additional available sensors were included in the neuromorphic control. This platform was used as a testbed to demonstrate that SNN architectures can control trajectories for this robot (Linares-Barranco et al., 2022). Specific data sets needed to train these neural networks can be collected with this platform.

This research was focused on improving and adapting a spike-based infrastructure for controlling the six joints of the ED-Scorbot and its use to collect trajectory datasets for machine learning. We allowed the flexibility to communicate the desired position of the joints from a computer (via a USB interface and a new Python interface) or from the output of a neuromorphic hardware, such as a SNN running in a Dynap-SE or Loihi platform, as is described in Donati et al. (2018) and Stagsted et al. (2020a).

In summary, this paper has the following list of contributions:

– Improved version of SPID controllers to access robot proprioceptive sensors, including filter circuits and required finite state machine (FSM) for extracting absolute joint positions from these sensors. This can be used by a controller algorithm, an Artificial Neural Network or SNN to implement trajectory controllers.
– A Python based interface that allows robot management through a Python script and / or through a graphical user interface (GUI), which can be run either locally or remotely, which gives access to an (embedded) Linux OS platform.
– An upgrade of the FPGA infrastructure from multi-Spartan platforms plus a dedicated computer, to a single system on chip (i.e., Zynq) platform without the need for an additional computer.
– Forward and inverse kinematics of the ED-Scorbot to calculate the robotic arm trajectories using Python scripting (code available upon request). These equations are used prior to computing the references for the spike-based control algorithm.
– Overall, a neuromorphic robotic platform available for the community to be used, for example, for SNN training and testing. Recorded trajectories are available.

The rest of the paper is structured as follows: Section 2 describes in detail the state of the art (materials) that supports this work, Section 3 provides all the implementations and improvements (ie. methods) that this work performs on the final neuromorphic robotic arm and its evaluation, Section 4 describes all the performed experiments and the obtained results Section 5 presents the conclusions of the paper.

2 Materials

This section provides details over the state of the art that supports this work. Firstly, the Spike-based PID (SPID) controller concept and its formulation is reviewed, offering the extraction of the PID constants (K_p, K_i, K_d). Then, a description of the robotic platform used in this work is provided.

2.1 Spike-based PID controller

Under the spike paradigm, a new kind of Proportional-Integrative-Derivative controller was designed: the SPID (Jimenez-Fernandez et al., 2012) controller. Both the reference input signal and the output signal to drive the motor are spike-based signals. The PFM modulation was used along the Spike-Signal-Processing (SSP) Building Blocks (Jimenez-Fernandez et al., 2010) to compute the intermediate results and to drive the motors.

The aforementioned input reference signal to the SPID controller can be provided by any neuromorphic system with a spiking firing rate as output, or it can be received as a digital value from a computer and converted into spikes (Gomez-Rodriguez et al., 2005). In Jimenez-Fernandez et al. (2012), the spiking input reference signal was used to control the speed of a motor, which was eventually converted to a two-independent-wheels, speed-controlled mobile robot. Therefore, the input frequency of spikes was converted to a fixed speed of a motor. In this work, the controller was modified to control the joint position of the Scorbot ER-VII robotic arm in a similar way as in Linares-Barranco et al. (2020).

The following lines summarize the SPID concept and its use in the ED-Scorbot. A PID controller continuously calculates an error value and applies a correction based on proportional, integral, and derivative terms (denoted P, I, and D, respectively), hence the name. In Jimenez-Fernandez et al. (2012), a set of SSP building blocks were developed taking into account the formulation of the Laplace domain (S-domain), as they obey the classic PID formulation. The basic building blocks are: the Spike-Generator, the Integrate-And-Generate (I&G) motor neuron model, the Hold-And-Fire (H&F) and the Spike-Expansor. These blocks are thoroughly explained in Jimenez-Fernandez et al. (2012); however, a brief explanation of how each of them work is provided next. The Spike-Generator receives a digital number (which is usually referred to as the *reference*) and outputs a spiking signal whose frequency changes depending on the *reference*. The Integrate-And-Generate module (I&G) performs a continuous integration of the received spikes, transferring the integration count of spikes to a bit-wise generator of spikes (Gomez-Rodriguez et al., 2005). It will generate a new signal of spikes whose frequency will be proportional to the

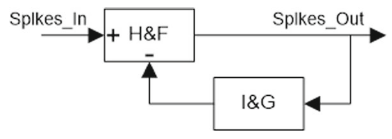


Fig. 1 Block diagram of the spike processing block known as "spike-Derivative". It consists of an Integrate and Generate block connected to a Hold and Fire for closing the loop. The result is that the output rate is the derivative of the input rate

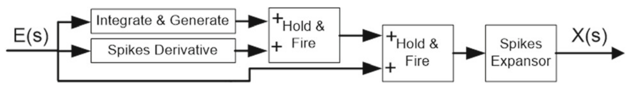


Fig. 2 Spike-based PID controller created with SSP building blocks

number of spikes counted. The SPID uses this block for the integral component. The role of the Hold-And-Fire module (H&F) is to merge spike streams adding or subtracting their frequencies by holding an incoming spike while it monitors the next input spike to decide whether output spikes must be produced or not depending on the polarity of both spikes. The right combination of these two blocks (see Fig. 1) allows extracting the behavior of the derivative component of the SPID. Finally, the Spike-Expansor block expands in time the spike signal generated by the controller. This step is necessary, since if the unexpanded spiking signal is used to feed the motors, these spikes will not be wide enough for the motors to recognise them as movement commands. The role of the Spike-Expansor in the controller is the proportional component. The controller shown in Fig. 2 was created using all these building blocks.

The transfer function in the S-domain of the full SPID controller follows Eq. (1). This SPID circuit is internally based on counters and comparators, avoiding the use of more complex operators, such as multipliers. This can be noticed from the formula, where K_i, K_d and K_p depend on the registers' length (NB), frequency dividers for the clock signals of the counters (FD) and the threshold for expanding an output spike in time (SW), apart from the main clock frequency (F_{CLK}) and the power supply (V_{PS}).

$$\begin{aligned} PID(S) &= \frac{X(S)}{E(S)} = \left(1 + \frac{K_i}{s} + \frac{s}{s + K_d}\right) \cdot K_p \\ &= \left(1+\frac{F_{CLK}}{2^{NB_i-1} \cdot FD_{GEN_i}} \cdot \frac{1}{s}+\frac{s}{s+\frac{F_{CLK}}{2^{NB_d-1} \cdot FD_{GEN_d}}}\right) \cdot \\ &\quad (SW+1) \cdot T_{CLK} \cdot V_{PS} \end{aligned} \quad (1)$$

For a robotic arm, this SPID speed controller needs a modification to be properly applied to each joint of a robotic arm for controlling the angles (position controller). Through the formulation of the forward kinematics, in order to obtain a Cartesian 3D coordinate of the end effector of the robot, in a way closer to biology, it is necessary to use stream of spikes is used for powering the motors, as a muscle is controlled in the same way by the nervous system. To foresee this, the speed controller has to be modified to become a position controller. This is done by inserting an additional Integrate-and-Generate block in the closed-loop, as in Linares-Barranco et al. (2020).

The SPID controller has important differences with respect to the classical discrete closed-loop PID. The classical PID is well established in the industry and can be found implemented in a wide variety of digital devices, such as Digital Signal Processors (DSP), microcontrollers, and embedded processors inside FPGAs. All these different types of devices need complex general purpose hardware to perform Multiplicative-and-Accumulative (MAC) operations, large buses, and shared resources, such as memory. Thus, it is not easy to implement a large number of digital real-time controllers running fully in parallel inside a single device. On the other hand, the spike-based PID-controller is very different from the discrete ones. It is also implemented in digital circuits, but it is similar to a continuous analog controller. SPID also has some benefits, e.g., a lower power consumption, since, in average, it will produce a lower commutation rate on the power stages. In addition, SPID controllers prevent intrinsic PID delays (Jimenez-Fernandez et al., 2012). However, the reduced latency and power consumption come at the expense of accuracy when reaching target points commanded by the controller, i.e., drift (Fig. 5 shows some of its effects). This drifting can be minimized by optimizing the K_p, K_i and K_d parameters, although this does not fall in the scope of this work.

2.2 Robotic platform

The Scorbot ER-VII is a six-axis arm robot with DC motors for moving axes. Each motor is provided with a dual optical encoder to enable the registration of axis movements. The SPID controllers are embedded now on the FPGA MPSoC hardware. A descriptive image of the operating range of the robot is shown in Fig. 3.

For controlling the robot axes, a set of six H-bridges (model BDMICRO REACTOR RX-50) controlled by an FPGA-based PCB is used. The high-power side of the circuitry is powered by a 12V-25A power supply. A general overview on the ED-Scorbot setup is shown in Fig. 4.

The SPID controllers were adjusted for the best response of the robot joint by joint. The final parameters of the controller, running with a 50MHz clock, and the values of the SPID constants are expressed in Table 2.

With these parameters in the SPID of the joints of the robot, different set points firing rate (from now on, spike reference) imply different angles in the joints. A spiking or spike reference is the set point that the controller receives as its

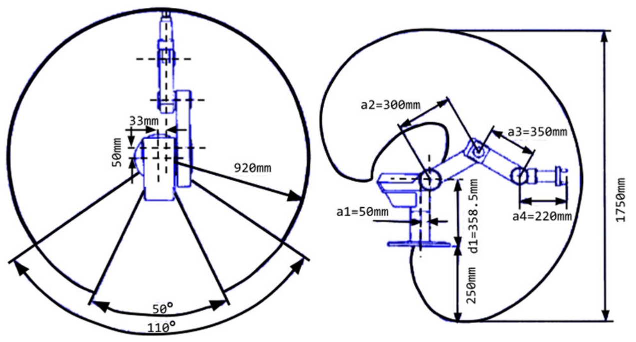


Fig. 3 Operating Range With Gripper Attached. Adapted from (Robotec, 1998)

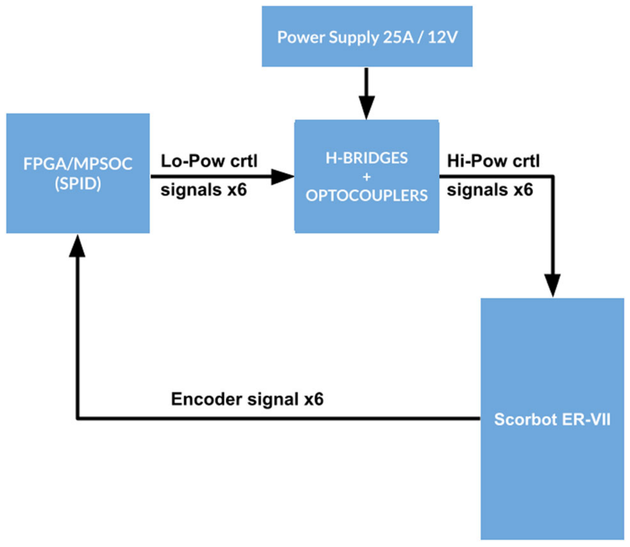


Fig. 4 General overview on the ED-Scorbot setup

input. This spiking signal can come either from a *Spike Generator*, such as in this work, or from another spiking source, such as an SNN or SNN accelerator (Linares-Barranco et al., 2022).

An SPID controller circuit, as explained in Sect. 2.1, was synthesized for the Spartan3 400 to measure the estimated power consumption with the Xpower tool for a 50% switching rate of the signals considering 16-bit width for all internal counters and buses. A classic PID controller was also synthesized, not based on spikes, with 16-bit register sizes [2] for the same FPGA. This PID uses pulse-width-modulation (PWM) to command the motors. The power measurements were performed in the same way as for SPID. Table 3 shows that the power benefit for one SPID motor controller is 3x with respect to an equivalent PID running at the same clock speed. Even though we have not yet tested the non-spiking PID controller and its consumption, the power saving of the SPID mainly depends on the reduction of switching signals. SPID reduces to the minimum the switching when the robot has reached its final position, but PID classic controllers maintain a constant signal switching, due to the use of PWM to drive the motors. SPID demonstrates other benefits, such as simplified operations (it does not require multipliers), and less use of resources.

[2] PID adapted from https://github.com/deepc94/pid-fpga-vhdl (accessed on 19-July-2021)

3 Methods

This section provides details of the improvements attained in the final neuromorphic robotic arm presented and evaluated in this work. Firstly, the direct and inverse kinematics of the arm are presented. Then, descriptions of the proprioceptive sensors are provided, and we explain how absolute position values per joint can be extracted from the encoder sensors of the motor, with both being new contributions in the SPID control in this work. Next, the novel Python software that supports both graphical-user-interface (GUI) and application-programming-interface (API), is described. Moreover, the improved implementation of the previous SPID from multi-FPGA plus a computer, to single Programmable System on a Chip (PSoC) Zynq is presented. Finally, this section provides a comprehensive comparison between ED-Scorbot and its predecessor: ED-BioRob.

3.1 Kinematics

The Scorbot ER-VII robotic arm has six DC motors, as it has been mentioned before. Since the last two of them are used to orientate and open/close the gripper located at the end-effector, the forward kinematics equations are computed using four joints. Table 4 shows the Denavit-Hartenberg parameters of the robotic arm. The values of the parameters are shown in Fig. 3.

The equations of the forward kinematics are shown in Eq. (2).

$$
\begin{aligned}
x &= a_4 c\theta_1 c\,(\theta_2\theta_3\theta_4) + a_3 c\theta_1 c\,(\theta_2\theta_3) + a_2 c\theta_1 c\theta_2 + a_1 c\theta_1 \\
y &= a_4 s\theta_1 c\,(\theta_2\theta_3\theta_4) + a_3 s\theta_1 c\,(\theta_2\theta_3) + a_2 s\theta_1 c\theta_2 + a_1 s\theta_1 \\
z &= a_4 s\,(\theta_2\theta_3\theta_4) - a_3 s\,(\theta_2\theta_3) - a_2 s\theta_2 + d_1
\end{aligned}
\tag{2}
$$

For the sake of clarity and brevity, we represent cos and sin operations with the letters c and s, respectively.

Regarding to the inverse kinematics problem, the general equations are shown in (3) (joint angles as a function of the target point in cartesian coordinates ($\vec{p} = (p_x, p_y, p_z)$). Since one of the experiments performed in this work includes an eight-shaped trajectory, in 2-dimensions, the equations

Table 2 K_p, K_i, K_d parameters of the first 4 SPID controllers of the ED-Scorbot joints and the closed-loop integrator (CL_i)

Joints	SW	K_p	NB_i	FD_{GEN_i}	K_i	NB_d	FD_{GEN_d}	K_d	NB_{CL_I}	$FD_{GEN_{CL_i}}$
J1	720	1.73e−4	18	1260	3.03e−1	22	512	4.65e−2	18	8
J2	370	8.90e−5	18	2674	1.43e−1	22	512	4.65e−2	18	2
J3	350	8.42e−5	18	3565	1.07e−1	22	512	4.65e−2	18	8
J4	202	4.87e−5	18	2122	1.80e−1	22	512	4.65e−2	18	1

NB is bit length, FD is frequency divider value and SW is spike width

Table 3 Power and resources estimation from ISE 14.7 Xilinx tools for SPID and PID with 16-bit register sizes

Controller	mW (at 120 MHz)	Slices	MULT18X18
PID	186	169	2
SPID	62	133	0

Table 4 Denavit-Hartenberg parameters of the robotic arm Scorbot ER-VII

Joint	θ_i	d_i	a_i	α_i
1	θ_1	d_1	a_1	$-\pi/2$
2	θ_2	0	a_2	0
3	θ_3	0	a_3	0
4	θ_4	0	a_4	0

consider that the wrist is fixed.

$$\begin{aligned}\theta_1 &= \arctan\left(\frac{p_y}{p_x}\right)\\ c(\theta_3) &= \frac{p_x^2 + p_y^2 + (p_z + d_1)^2 - a_2^2 - a_3^2 - a_4^2}{2a_2a_3a_4}\\ \theta_3 &= \arctan\left(\frac{\pm\sqrt{1 - c^2(\theta_3)}}{c(\theta_3)}\right)\\ \theta_2 &= \arctan\left(\frac{(p_z + d_1)}{\pm\sqrt{p_x^2 + p_y^2}}\right)\\ &\quad - \arctan\left(\frac{(a_3 + a_4)s(\theta_3)}{a_2 + (a_3 + a_4)c(\theta_3)}\right)\end{aligned} \tag{3}$$

3.2 Proprioceptive sensors

Proprioceptive sensors measure internal values of the system or robot. The Scorbot ER-VII has a different set of proprioceptive sensors with respect to other event-based robots (i.e., ED-BioRob (Linares-Barranco et al., 2020)). The Scorbot does not provide joint angle sensors, but it includes, per joint, three microswitches (one for the home position and two for the joint limits) and a dual channel optical encoder attached to the motor. Additional filtering circuits were developed and included to solve both the bouncing effect of the microswitches (2 ms max) and sporadic glitches of 20–40 ns in the wiring due to electromagnetic noise. Optical encoder signals are converted into spikes to obtain the feed-back signal that is connected to the I&G block of the closed-loop. Since the encoder has two channels (A and B) and these are used to know the direction of the joint's movement, this information is used to extract the right polarity of the spikes. A positive or negative spike ($spike_p$ or $spike_n$) is generated for each change in either channel A or B of the encoder for the corresponding movement direction of the joint. Therefore, there is actually no sensor for the global joints position. Nevertheless, in this spike-based neuromorphic controller integration for the ED-Scorbot, an additional circuit was implemented in the FPGA per joint, consisting of an 18-bit counter whose increment or decrement signal is connected to encoders $spike_p$ and $spike_n$ respectively. In this way, after homing the robot, these counters offer a global position of the robot that can be used to estimate the orientation (in degrees) of each joint.

The home position of the Scorbot robotic arm is not a single point but a range, and the homing procedure of the robot must be performed before using it. The home microswitch of a joint is placed in a fixed position of the joint path. When the joint crosses this home position, the micro-switch is active for a particular range. Since these ranges are not small enough for all joints, we have considered as the home position the point where the switch outputs an edge from 0 to 1 when the joint is moving from the negative limit to the positive limit. Therefore, a routine in the Python library has been coded to manage this operation by sending each joint (one at a time) to its negative limit. Then, this routine keeps moving the joint in the positive direction until an edge in the home signal is detected. Then the 18-bit counter is reset to its middle value 0x20000. Furthermore, this procedure of resetting the counters can be enabled to work only once when homing the robot, or anytime. This way lets the robot to automatically fix possible drift or imprecise target position achievements due to joint force issues (ie. joint 3, see Fig. 9 bottom-left). The software only receives 16-bits from these counters per joint, discarding the two less significant bits (thus, the home value is 32768 in decimals). This is done in this way because of two reasons: first, most interfaces between logic and processors are words whose length is multiple of a byte; and second, each

Table 5 Characterization of joints' references

Joint	Ticks/degree	Bounds	Op. range
1	128	[11771, 51158]	$[-155\circ, 155\circ]$
2	152	[18478, 44353]	$[-85\circ, 85\circ]$
3	133	[17583, 47591]	$[-112.5\circ, 112.5\circ]$
4	80	[25477, 39797]	$[-90\circ, 90\circ]$

edge of the encoder represents an increment (or decrement) in the counter, being four steps for a period of the encoder. Table 5 shows the measured limits of these sensors per joint in the ED-Scorbot after the homing procedure.

3.3 Python software interface

The managing software for this implementation of the SPID controller was originally coded in Java, making use of the jAER software framework.[3] Through this application, one could command the robot to move, change its internal registers in order to change the K_p, K_i, K_d constants and even program various algorithms, such as home-searching or different trajectories. The downside, however, was that all this had to be done from a graphical interface and everything was also subject to the framework's visual programming structure, oriented to event-based sensors and their processing.

Therefore, we decided to re-code the old Java GUI software in Python, supporting both, a GUI, and an API. This decision was based on two main reasons. Firstly, Python is the main language in the Machine Learning field. Secondly, the Robotic Operating System (ROS) supports Python and provides tutorials and examples to facilitate the access to new researchers. ROS support will be included in this platform in the future. Furthermore, this also allowed us to provide remote access to the controller of the robot. Therefore, the managing software of this work was firstly written in pure Python, adapting the previous one and maintaining the same functionalities, while also adding new features such as being able to execute scripts that perform one (or more) algorithms, as mentioned before. This is quite an advantageous characteristic, as it was not viable to use it through graphical interface neither for remote access nor for automated learning of a neural network, for example. The new tool was named Py-EDScorbotTool.[4] As an improvement, we also developed a C/C++ managing software to be run on the PSoC-based infrastructure, due to the PS minimal OS of the Zynq platform that communicates directly over the AXI bus with the PL. This would significantly improve the performance of the platform due to latency reduction (more details in next subsection).

Therefore, for the multi-FPGA-based infrastructure, there is a dedicated server connected to the robot's SPID controller that includes the Python tool and acts as an interface for everyone who wants to use it, including the GUI, as the dedicated server is also connected to the Internet. Similarly, for the PSoC, the Python API is remotely available, but the GUI is executed locally. This API, in this case, keeps using C/C++ modules to interact with the robot more quickly. Py-EDScorbot provides everything any user would need to get started, and the tool also provides information on how to use it. However, due to safety and scheduling constraints, the authors must be contacted in order to make an appointment to use the platform.

3.4 From multi-FPGA to single-PSoC

The first attempt to convert the Scorbot into a neuromorphic ED-Scorbot was based on the same concept as in the ED-BioRob (Linares-Barranco et al., 2020), but taking into account the particularities of this robot. This infrastructure lacks several main aspects: (1) Spartan FPGAs are deprecated, difficult to acquire and not supported by new development tools; (2) the use of multiple FPGAs due to resource scarcity induces sporadic communication errors, longer latency and complexity in the incorporation of new features; (3) a dedicated computer with a USB connection is needed to configure and use the robot; and (4) the overall power consumption increases.

The next step in the building of FPGA based architectures for several different tasks, including controlling a robotic arm, is to integrate a PSoC to push the limits of the power consumption and latency to the edge while still delivering a proper solution. In this work, a Xilinx Zynq-7000® All Programmable SoC Mini-Module Plus 7Z100 (MMP) was used. This device is made up of two differentiable parts: the Processing System (PS) and the Programmable Logic (PL). PS functions through one ARM processor in which a device-tailored, minimal Linux distribution can be run. PL holds all FPGA resources of the platform, as it contains a Kintex 7 FPGA, which provides enough resources to hold both old Spartan projects as well as future requirements.

A simplified block diagram of this implemented infrastructure is shown in Fig. 6. As can be observed, the PL holds the controller fully, whereas the previous implementation needed two separate Spartan boards to allocate all the sufficient resources. On the other hand, the PS runs a Linux OS that allows it to have seamless access to standard ports and protocols such as Ethernet or USB. This still allows remote access to the platform, as the Linux OS will provide SSH access. Finally, PL and PS communicate through a standard Advanced eXtensible Interface (AXI) bus, which constitutes

[3] https://github.com/SensorsINI/jaer (accessed on 26/Feb/2021)

[4] https://github.com/RTC-research-group/Py-EDScorbotTool (accessed on 28/Feb/2021)

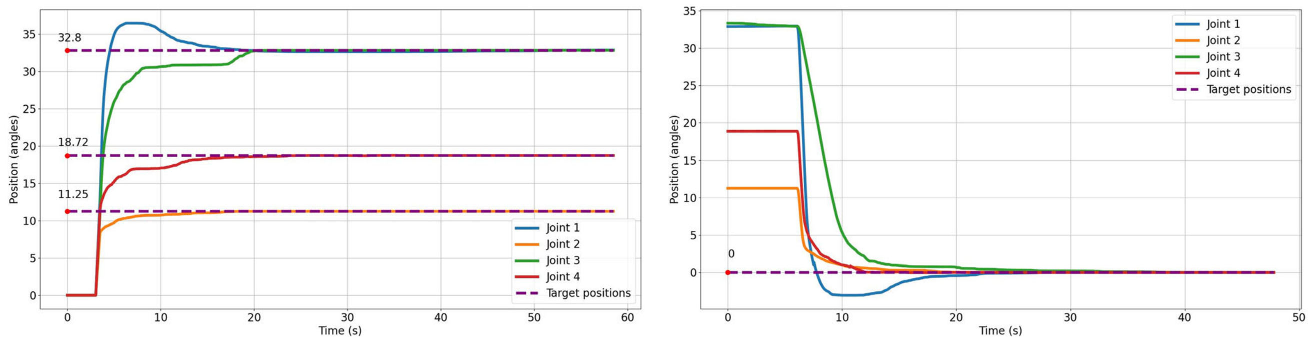


Fig. 5 Drift and wait time to reach a target position accurately. Each joint is commanded to move with the same digital reference, which produces a different angle in each joint as they do not have the same characteristics. No joint will be able to reach its target position accurately if not enough time is given

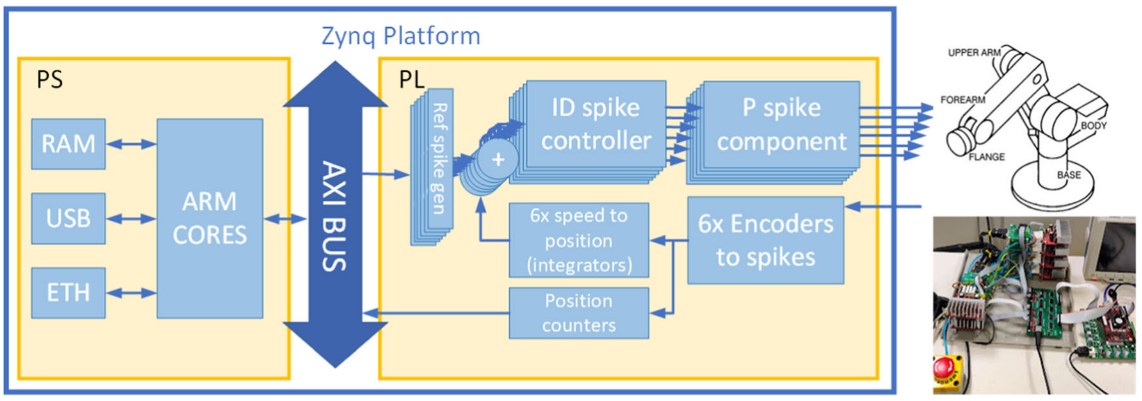


Fig. 6 Hardware platform block diagram (left) for 6× DoF spike PID control and photograph of the zynq platform connected to the robot power stage (right)

Table 6 Dynamic power and latency metrics for PL (Spartan & zynq) and PS (zynq)

System	Dyn. (PL) (mW)	Dyn. (PS)	Latency (ms)
Zynq	57	1584 mW	6.5
Spartan3/Spartan6	157	N/A	40

a significant improvement in latency over the SPI bus used in the previous platform. This is an improvement to the previous Spartan-based setup to maintain it up-to-date with current technologies.

Table 6 shows the response speed, the estimated power consumption, and the latency of both infrastructures (without considering the high-voltage power needed for driving the robot motors). Latency was significantly reduced, while PL consumption is also significantly lower for the new system than for the old one. In addition to this, the power consumption of the PS is almost null compared to a dedicated server, which would normally include a power supply of at least 300W (450W in our scenario).

3.5 ED-Scorbot VII versus ED-BioRob

The ED-Biorob is a very light robot, with joints damped with a system of ropes and pulleys. This caused overoscillations when commanding a position, although it posed an advantage for working together with humans, as its general applied force is lower than the force applied to an industrial robot.

The ED-Scorbot has its joints connected to the motors by belts, with the joints being more rigid. Unlike the ED-BioRob, the ED-Scorbot only has digital optical encoders that allow you knowing which the direction in which each motor is moving and how many degrees it is moving in that direction. However, they do not show the absolute position of a joint, unlike the ED-BioRob, which provides other absolute sensors. An estimated position can be calculated if a homing process is performed in advance. In such process, a limit switch is used to know the "home" position of that joint. Once the home process has been performed, the estimated position is obtained by counting the degrees that the associated motor moves from the home position, thus integrating the movements detected by the encoders with respect to the joint's home position. This estimation is possible due to the characterization described in Sect. 4.1. The ED-Scorbot also has counterweights and larger dimensions. Its movement is faster than that of the ED-BioRob, but its motors' energy consumption is also higher.

4 Results

In this section we present experimental results of the platform's performance. The characterization of the four first joints used for trajectory path is explained. Then, two examples of trajectories are presented and discussed. Finally, we introduce the mechanisms for collecting useful datasets for machine learning algorithms that could learn to produce trajectories with this robot.

4.1 ED-Scorbot joints characterization

For a correct use of the robot, a relation between the spike input reference and the angle of a joint must be established. To this end, a featuring process for the first four axes was performed (the fifth axis was not relevant for the preliminary study and the sixth axis, which corresponds to the grip, will be used in the future for grasping objects). The process consisted of taking samples of a triplet (spike reference, encoder counter value, axis angle), with the angle being related to a home (origin) position in the axis. To improve the rightness of the process, two batches of samples were done, one in the direction increasing the spike reference and the other batch in the opposite direction.

For taking samples in Axis 1, a flexible metric tape was adhered to the base plate of the axis to serve as an angle scale. The angle was indicated by a pencil properly placed at the end of the arm grip, as can be seen in Fig. 7 top-left.

For the other axes, a photo camera placed orthogonal to the axis was used to take snapshots. The angles were measured by fitting positioning boxes (red rectangles in Fig. 7) over the arm segment moved by the axis. The angle can be assessed obtaining the rotation angle of the box, related to another reference (which could be a horizontal plane or the positioning box of another segment of the arm). Figure 7 also shows three snapshots taken for measuring the angles of Joints 2, 3 and 4.

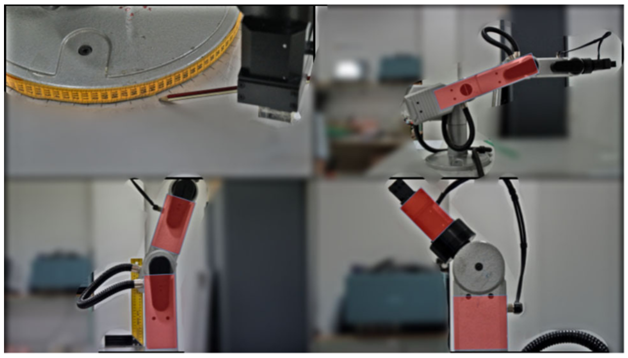


Fig. 7 Joints characterisation scenario photographs. Top-left is Joint 1, top-right is Joint 2, bottom left is Joint 3 and bottom right is Joint 4

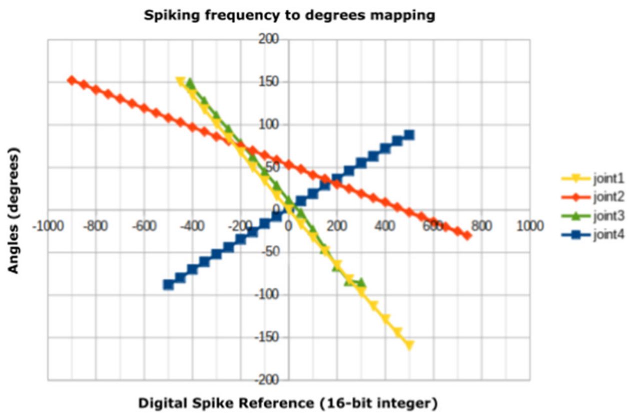


Fig. 8 Characterisation of Joints 1–4

The measured process concluded with a data processing step to generate the desired relation, expressed graphically in Fig. 8. The spike reference (SR), shown in the horizontal axis, is a digital value that represents the spiking frequency ($rate$) of the spike generator as the SPID input reference ($rate = SR \times F_{CLK}/2^{(n-1)} = 1525.88xSRtextspikes/s$ for 50MHz and 16-bits) Linares-Barranco et al. (2020). The angle is in the vertical axis.

As can be seen in Fig. 8, the right part of the Joint 3 curve is not as linear as the left part, due to the fact that the motor of that joint has malfunctions (due to its age). Thus, there is an imprecise relation between position and spike references, which must be compensated in higher level processing layers. To observe that imprecision, a new batch of measurements were taken for each axis.

Figure 9 shows normalized recorded and commanded positions (scaled between 0 and 1 using Eq. 5) for a geometrical comparison, after homing the robot and reseting the counters only once after the homing procedure. It measures the joint's position counters (blue) with respect to the commanded spike input reference (red). Y-axis shows the normalized angle of the spiking reference commanded (blue) and the position counters (red). X-axis represents the sample

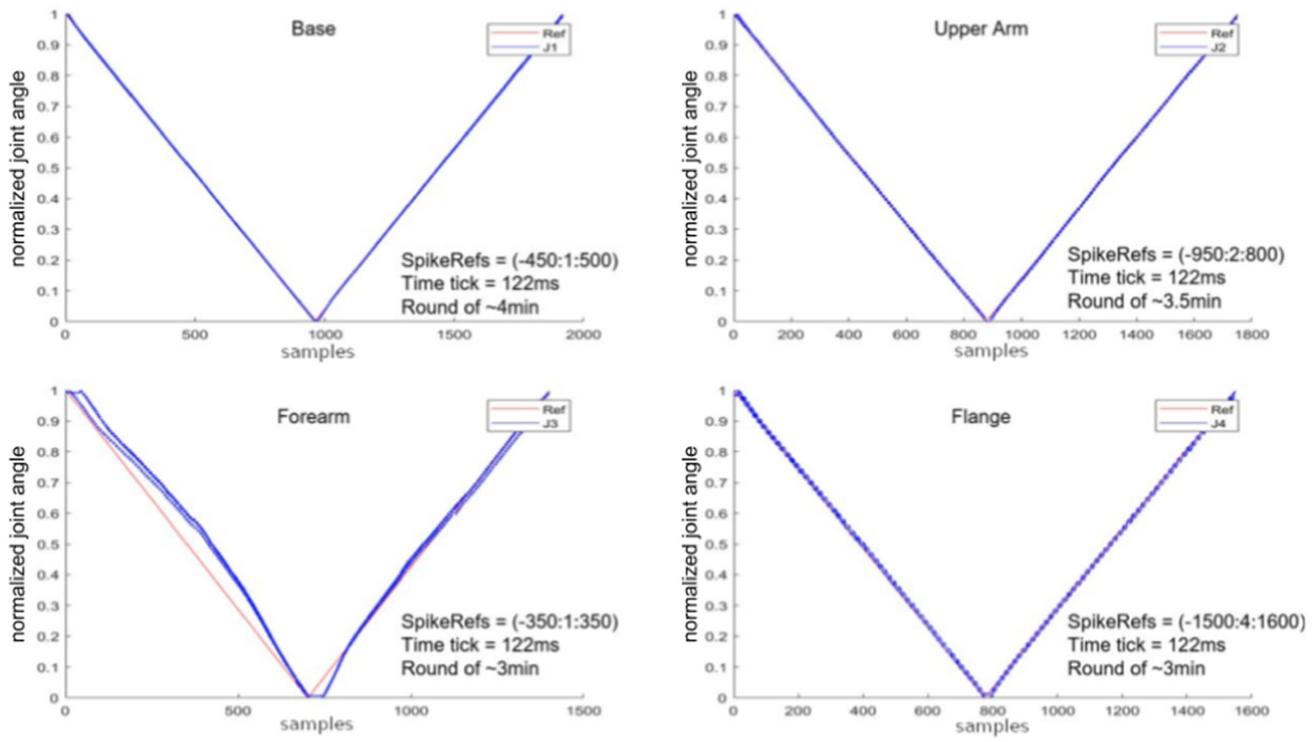


Fig. 9 Normalized joint angle and spiking reference for a 5 iteration movement over the joints limits for a digital spike reference for different steps. For J1 (base) and J3 (forearm), spike references have a step increment of 1, while J2 (upper arm) has a step of 2 and J4 (flange) a step of 4. Each command is sent from the software to the robot with a latency of 122ms per command

Table 7 RMSE per joint for each iteration

	It. 1	It. 2	It. 3	It. 4	It. 5
Joint 1	0.0041	0.0042	0.0042	0.0041	0.0042
Joint 2	0.0064	0.0065	0.0065	0.0065	0.0065
Joint 3	0.0430	0.0534	0.0541	0.0546	0.0545
Joint 4	0.0108	0.0111	0.0110	0.0110	0.0108

time, where each sample is captured with a time period specified in the figure ($Time\ tick = 122$ms/sample). Each joint was measured with independent experiments for 3–4 min, commanding a new $SpikeRef$ every $Time\ tick$ with the specified $SpikeRef$ step (1 for Joints 1 and 3, 2 for joint 2 and 4 for joint 4). The SPID parameters used in this configuration imply different digital spike references per joint as is shown in the figure. This experiment confirms the previous results for Joint 3, where it can be seen that the upward movement (left part of the Joint 3 curve) has a worse behavior than the downward movement.

Table 7 shows the root-mean-square error (RMSE) (Eq. 4) of normalized data of each joint in all five iterations shown in Fig. 9. In this equation, f_c is the value commanded to the arm and f_r is the obtained value. According to this result, the closer we get to 0, the lower the difference between the commanded position and the obtained position.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}\left(f_{c_i} - f_{r_i}\right)^2}{N}} \tag{4}$$

From the normalized commanded and collected data (set $\mathbf{x}'$), we calculated the error using the RMSE formula (4).

$$x_i' = \frac{x_i - min(\mathbf{x})}{max(\mathbf{x}) - min(\mathbf{x})} \tag{5}$$

Where $\mathbf{x}$ denotes the whole data collection and $min(\mathbf{x})$ and $max(\mathbf{x})$ are the minimum and maximum values of the set $\mathbf{x}$.

4.2 Two case-studied trajectories

This section presents results from two experimental trajectories: one where all the joints move at the same time for the same range of digital spike references drawing a curve in the space, and a second one where the robot is drawing an "eight" in space. A short video for each experiment can be accessed in the Py-EDScorbotTool git repository previously referenced. In all the experiments carried out in this research, hand-set PID constants were used.

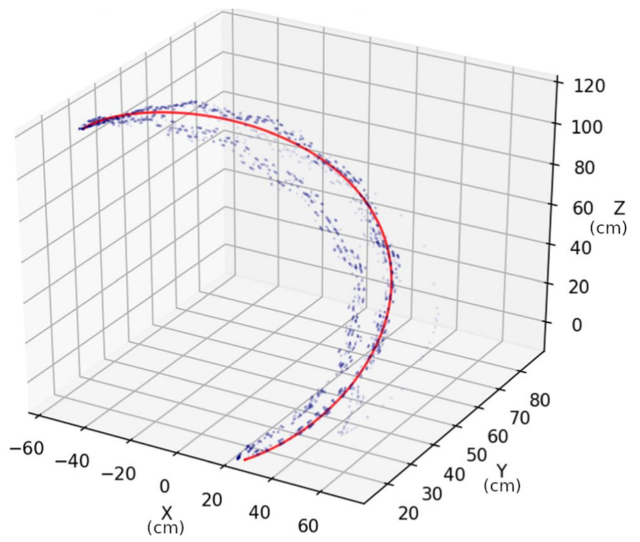


Fig. 10 Curve experiment for 10 iterations. Blue dots are the 3D representation of the measured joint angles computed by the forward kinematics. Red dots correspond to the commanded angles converted to Euclidean coordinates in cm

4.2.1 Curve experiment

In this experiment, a script for commanding the same digital spike reference to each SPID input of the 4 considered joints was developed. These spike references where in the range of -200 to 200. Therefore, the angles produced for each joint have different ranges according to the characterization presented in previous sections. With this range, starting at the home position (and reseting the position counters to 0×20000 only once after homing procedure), the robot reproduces an arc that starts in the desk at its right side and finishes on top of the robot, then, the robot comes back to the desk and the movement is repeated 5 times, as can be seen in Fig. 10. The script increments the digital spike reference (from -200 to 200) one by one and sends the commands to the robot joint at its maximum speed (125 ms per increment), which implies 100 s for each of the 5 iterations. The blue dots (measured positions) do not precisely correspond with the red dots (commanded position) due to latencies of the SPID in stabilizing at targeting positions during the trajectory. The mean error measured calculating the distance between the target point and the measured point for the whole experiment is 10.38 cm. The use of a SNN properly trained for dynamically updating the controller parameters may improve these results.

4.2.2 The fast 'eight' experiment

This experiment shows how the robotic arm is following a two-dimension trajectory: an image of the number eight. The trajectory was extracted using Eq. (6), which correspond to a semicircle, where r is the radius, θ is the angle in radians from 0 to π and (h, k) is the center of the circumference. Thus, combining and concatenating four semicircles the 'eight' approximation can be drawn for this experiment. Joints 1 and 2 are used and joints 3 and 4 are left in their home values. The Python script for this experiment allows representing the trajectory with configurable scales and sample points. The experiment was continuously repeated 5 times and recorded on a log file. Each iteration consisted of 80 points commanded in 6.25 s, which corresponds to one command every 125 ms to move from one point to the next, which in turn is the maximum speed for commanding the robot. Figure 11 represents with blue dots the 3D Euclidean space position of the end-effector of the robot for the different iterations taking the angles from the log file and converting them into (x, y, z) with the previous kinematics equations. Red lines correspond to the commanded trajectory. Since the highest speed available was used in this experiment to command the positions, the error in the measurements is also the highest. Current PID constants and motor speed do not allow the robot to reach the commanded positions on time. In this case, the average error in this trajectory is 6.80 cm.

$$\begin{aligned} x &= r \cdot \cos\theta + h \\ y &= r \cdot \sin\theta + k \end{aligned} \tag{6}$$

4.3 Datasets generation

This robot is being used as testing platform for neuromorphic learning architectures under an EU project. One of the aims of these spiking architectures is to modify the PID constants of the robot in a dynamic way for fixing the observed problems of non-linearity (i.e., the robot's Joint-3), or due to object grasping with different weights. Some of these architectures require the use of datasets for their training phases. A Python script can always be used with the robot for collecting required datasets using available features. The Python library currently in use, allows storing text files with the robot parameters' values online (while it performs the programmed trajectory). In the results section, these text files were used to extract the figures. While the main software is running, it constantly saves the latest commanded position of the joints, the current value of the measured joint position and a timestamp. The maximum speed is 1 measurement of the four joints and its log file writes every 122 ms. The trajectories dataset[5] is complemented with recorded MP4 videos of three cameras installed on the X, Y and Z axes. Finally, the spiking activity of the SPID controllers is recorded using an USBAERmini2 board (Berner et al., 2007), and stored in an AEDAT file as a sequence of (x, y) addresses with the

[5] https://github.com/RTC-research-group/ED-Scorbot.

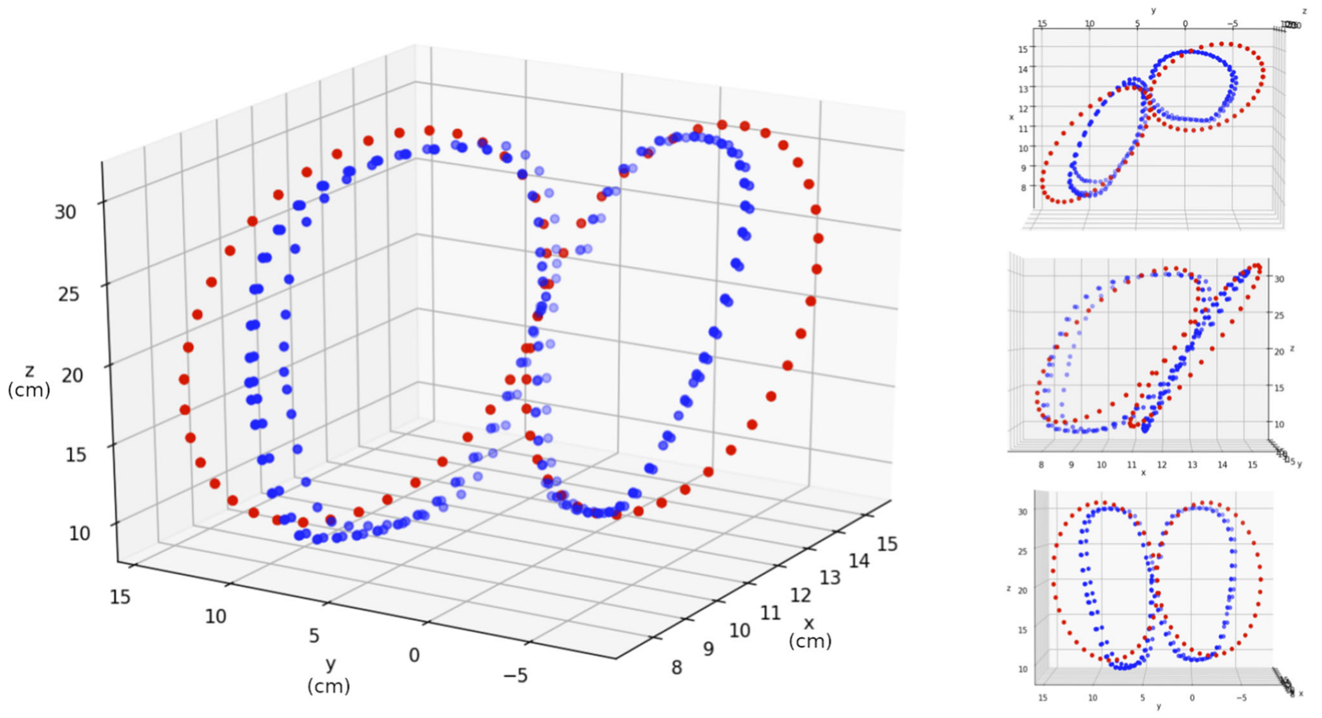


Fig. 11 “Eight” experiment for 5 iterations. Blue dots are the measured joint angles forward kinematics representation. Red dots corresponds to the commanded angles converted to Euclidean coordinates. Side figures show the experiment as if the observer were in front of each axis

timestamps with which these spikes appeared. Table 8 shows the format of the recorded events while a trajectory is executed in the robot. $Source_1 - Source_0$ values identify the source of the captured spike, being “00” for the output of the spike generator that converts the digital spike-reference to spiking activity; “01” for the SPID output; “10” for the SPID input; and “11” for the integral of the spiking encoder activity. $Joint_2 - Joint_0$ encode the joint number; and Pol for the polarity of the captured spike. Figure 12 shows recorded inter-spike-interval (ISI) events for Joint 2 (upper arm) SPID while producing a simple trajectory consisting in sending the base, upper arm, forearm and flange from the digital spiking reference −200 to 200 in steps of 10 with the minimum wait between steps (122 ms) repetitively (5 times). SPID spiking output changes its ISI time depending on the speed of the movement when it reaches one side or the other of the repetitive movement. It can be seen that the activity of the integration of the motor encoders (aka position) obeys the SPID output activity with a delay. This delay is caused by the dynamics of the motors and the robot joints. The figure shows with different colours positive and negative spikes for these two signals: SPID output and the integral of the spiking encoder signal.

Table 8 Address-Event-Representation of SPID internal activity monitoring

Bit_5	Bit_4	Bit_3	Bit_2	Bit_1	Bit_0
$Source_1$	$Source_0$	$Joint_2$	$Joint_1$	$Joint_0$	Pol

5 Conclusions

In this paper, an application of a Spike-based Proportional-Integral-Derivative position controller to a Scorbot ER-VII robotic arm is presented. This design was further tested and simulated. The Spike-based PID controller was improved and applied to each joint of the robot. The robotic arm platform was improved to allow access to its proprioceptive sensors. Remote access through SSH with webcams and a Python-based interface was added to the robotic platform to provide remote control over the robotic arm. Two infrastructures for the use of a SPID controller are presented: one that uses deprecated Xilinx Spartan FPGAs and an update of the first one that uses a more recently developed Xilinx Zynq MPSoC. In the first infrastructure, two Spartan boards are required to fit the whole controller, and they also require external

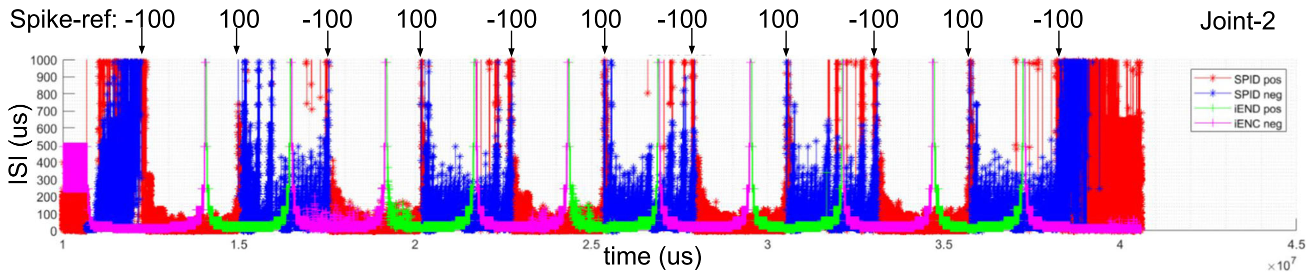


Fig. 12 Monitored spiking activity of the SPID output and the integral of the spiking encoder for Joint 2 while the ED-Scorbot executes a trajectory where all joints are moving from digital spiking reference −100 to 100, with ref 0 being the HOME position

support through the use of a dedicated PC/server that allows for USB access. However, in the second infrastructure, the MPSoC's PL is capable of holding the whole controller in one chip, and the PS serves as a dedicated server, but with much less power consumption. Inter-FPGA communication was also removed in the MPSoC; therefore, latency improved significantly. The control system presented in this paper uses spiking neurons that are tuned to achieve the desired PID control. However, our system lacks ground-truth sensors to determine the position of each joint, while other works do have them (Linares-Barranco et al., 2020). Furthermore, this infrastructure can be used for collecting trajectories dataset with video captures and spiking information for potential training of machine learning algorithms.

Acknowledgements This research was partially supported by the Spanish grant (with support from the European Regional Development Fund) MINDROB (PID2019- 105556GB- C33/ AEI/ 10. 13039/ 501100011033) and by the CHIST-ERA H2020 grant SMALL (CHIST-ERA-18-ACAI-004, PCI2019-111841-2/AEI/10.13039/501100011033).

Funding Funding for open access publishing: Universidad de Sevilla/ CBUA

Conflict of interest The authors have no conflict of interest to declare that are relevant to the content of this article.

References

Benjamin, B. V., Gao, P., McQuinn, E., Choudhary, S., et al. (2014). Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations. *Proceedings of the IEEE, 102*(5), 699–716.

Berner, R., Delbruck, T., Civit-Balcells, A., & Linares-Barranco, A. (2007). A 5 meps $100 usb2.0 address-event monitor-sequencer interface. In *2007 IEEE international symposium on circuits and systems,* (pp. 2451–2454).

Blum, H., Dietmüller, A., Milde, M., Conradt, J., et al. (2017). A neuromorphic controller for a robotic vehicle equipped with a dynamic vision sensor. In: *Robotics Science and Systems, RSS 2017, Berlin, Germany. Proceedings of Robotics: Science and Systems*.

Calimera, A., & Macii, E. (2013). The human brain project and neuromorphic computing. *Functional Neurology, 28*(3), 191–196.

Cassidy, A. S., Merolla, P., Arthur, J. V., Esser, S. K., et al. (2013). Cognitive computing building block: A versatile and efficient digital neuron model for neurosynaptic cores. In: *The 2013 international joint conference on neural networks (IJCNN),* (pp. 1–10).

Chan, V., Liu, S., & van Schaik, A. (2007). Aer ear: A matched silicon cochlea pair with address event representation interface. *IEEE Transactions on Circuits and Systems I: Regular Papers, 54*(1), 48–59.

Chicca, E., Stefanini, F., Bartolozzi, C., & Indiveri, G. (2014). Neuromorphic electronic circuits for building autonomous cognitive systems. *Proceedings of the IEEE, 102*(9), 1367–1388.

Davies, M., Srinivasa, N., Lin, T., Chinya, G., et al. (2018). Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro, 38*(1), 82–99.

Denk, C., Llobet-Blandino, F., Galluppi, F., Plana, L., et al. (2013). Real-time interface board for closed-loop robotic tasks on the spinnaker neural computing system. In: *Artificial neural networks and machine learning (ICANN 2013),* (pp. 467–474). Springer Berlin Heidelberg.

DeWolf, T., Stewart, T. C., 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.

Dominguez-Morales, M., Jimenez-Fernandez, A., Paz, R., Linares-Barranco, A., et al. (2011). An AER to can bridge for spike-based robot control. In *Advances in Computational Intelligence,* (pp. 124–132). Springer Berlin Heidelberg.

Donati, E., Perez-Peña, F., Bartolozzi, C., Indiveri, G., and Chicca, E. (2018). Open-loop neuromorphic controller implemented on VLSI devices. In *2018 7th IEEE International conference on biomedical robotics and biomechatronics (Biorob),* (pp. 827–832). IEEE.

Furber, S. B., Galluppi, F., Temple, S., & Plana, L. A. (2014). The spinnaker project. *Proceedings of the IEEE, 102*(5), 652–665.

Gallego, G., Delbruck, T., Orchard, G. M., Bartolozzi, C., et al. (2020). Event-based vision: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 44*, 154–180.

Galluppi, F., Denk, C., Meiner, M. C., Stewart, T. C., et al. (2014). Event-based neural computing on an autonomous mobile platform. In: *2014 IEEE international conference on robotics and automation (ICRA),* (pp. 2862–2867).

Gomez-Rodriguez, F., Paz, R., Miro, L., Linares-Barranco, A., et al. (2005). Two hardware implementations of the exhaustive synthetic AER generation method. *Computational Intelligence and Bioinspired Systems* (pp. 534–540). Berlin Heidelberg: Springer.

Gómez-Rodríguez, F., Jiménez-Fernández, A., Pérez-Peña, F., Miró, L., et al. (2016). Ed-scorbot: A robotic test-bed framework for FPGA-based neuromorphic systems. In *2016 6th IEEE international conference on biomedical robotics and biomechatronics (BioRob),* (pp. 237–242).

Harkin, J., Morgan, F., Hall, S., Dudek, P., Dowrick, T., & McDaid, L. (2008). Reconfigurable platforms and the challenges for large-scale implementations of spiking neural networks. In *2008 international conference on field programmable logic and applications,* (pp. 483–486). IEEE.

Jimenez- Fernandez, A., Linares-Barranco, A., Paz-Vicente, R., Jimenez, G., & Civit, A. (2010). Building blocks for spikes signals processing. In *The 2010 international joint conference on neural networks (IJCNN)*.

Jimenez-Fernandez, A., Jimenez-Moreno, G., Linares-Barranco, A., Dominguez-Morales, M., et al. (2012). A neuro-inspired spike-based PID motor controller for multi-motor robots with low cost FPGAS. *Sensors, 12*(4), 3831–3856.

Koickal, T. J., Hamilton, A., Tan, S. L., Covington, J. A., et al. (2007). Analog VLSI circuit implementation of an adaptive neuromorphic olfaction chip. *IEEE Transactions on Circuits and Systems I: Regular Papers, 54*(1), 60–73.

Lichtsteiner, P., Posch, C., & Delbruck, T. (2008). A 128$\times$ 128 120 db 15 μs latency asynchronous temporal contrast vision sensor. *IEEE Journal of Solid-State Circuits, 43*(2), 566–576.

Linares-Barranco, A., Perez-Pena, F., Jimenez-Fernandez, A., & Chicca, E. (2020). Ed-biorob: A neuromorphic robotic arm with FPGA-based infrastructure for bio-inspired spiking motor controllers. *Frontiers in Neurorobotics, 14*, 96.

Linares-Barranco, A., Pinero-Fuentes, E., Canas-Moreno, S., Rios-Navarro, A., Maryada, E., C. Wu, J. Z., Zendrikov, D., & Indiveri, G. (2022). Towards hardware implementation of WTA for CPG-based control of a spiking robotic arm. In *2022 IEEE international symposium on circuits and systems,* (pp. 1–4).

Maguire, L., McGinnity, T., Glackin, B., Ghani, A., Belatreche, A., & Harkin, J. (2007). Challenges for large-scale implementations of spiking neural networks on FPGAS. *Neurocomputing, 71*(1), 13–29.

Milde, M., Blum, H., Dietmuller, A., Sumislawska, D., et al. (2017). Obstacle avoidance and target acquisition for robot navigation using a mixed signal analog/digital neuromorphic processing system. *Frontiers in Neurorobotics, 11*, 28.

Moradi, S., Qiao, N., Stefanini, F., & Indiveri, G. (2018). A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (dynaps). *IEEE Transactions on Biomedical Circuits and Systems, 12*(1), 106–122.

Perez-Pena, F., Morgado-Estevez, A., Linares-Barranco, A., Jimenez-Fernandez, A., et al. (2013). Neuro-inspired spike-based motion: From dynamic vision sensor to robot motor open-loop control through spike-vite. *Sensors, 13*(11), 15805–15832.

Qiao, N., Mostafa, H., Corradi, F., Osswald, M., et al. (2015). A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128 k synapses. *Frontiers in Neuroscience, 9*, 141.

Robotec, E. (1998). *Scorbot er-vii user manual*. Eshed Robotec Limited.

Rocke, P., McGinley, B., Maher, J., Morgan, F., & Harkin, J. (2008). Investigating the suitability of FPAAS for evolved hardware spiking neural networks. In *Evolvable systems: From biology to hardware,* (pp. 118–129). Springer Berlin Heidelberg.

Schemmel, J., Briiderle, D., Griibl, A., Hock, M., Meier, K., & Millner, S. (2010). A wafer-scale neuromorphic hardware system for large-scale neural modeling. In *Proceedings of 2010 IEEE international symposium on circuits and systems,* (pp. 1947–1950).

Schemmel, J., Fieres, J., & Meier, K. (2008). Wafer-scale integration of analog neural networks. In *2008 IEEE international joint conference on neural networks,* (pp. 431–438).

Selow, R., Lopes, H. S., & Lima, C. R. E. (2009). A comparison of FPGA and FPAA technologies for a signal processing application. In *2009 international conference on field programmable logic and applications,* (pp. 230–235). IEEE.

Serrano-Gotarredona, T., & Linares-Barranco, B. (2013). A 128 $\times$128 1.5

Sivilotti, M. A. (1992). *Wiring considerations in analog VLSI systems, with application to field-programmable networks,* PhD thesis, California Institute of Technology, USA.

Stagsted, R., Vitale, A., Binz, J., Renner, A., et al. (2020a). Towards neuromorphic control: A spiking neural network based PID controller for UAV. In *Robotics: science and systems 2020,* RSS.

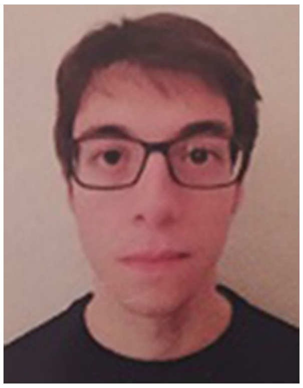
Stagsted, R. K., Vitale, A., Renner, A., Larsen, L. B., et al. (2020b). Event-based pid controller fully realized in neuromorphic hardware: A one DOF study. In *2020 IEEE/RSJ international conference on intelligent robots and systems (IROS),*(pp. 10939–10944).

Stroobants, S., Dupeyroux, J., & de Croon, G. (2021). Design and implementation of a parsimonious neuromorphic PID for onboard altitude control for MAVS using neuromorphic processors. arXiv preprint arXiv:2109.10199.

Zaidel, Y., Shalumov, A., Volinski, A., Supic, L., & Ezra Tsur, E. (2021). Neuromorphic NEF-based inverse kinematics and PID control. *Frontiers in Neurorobotics, 15*, 2.

Zamarreno-Ramos, C., Linares-Barranco, A., Serrano-Gotarredona, T., & Linares-Barranco, B. (2013). Multicasting mesh AER: A scalable assembly approach for reconfigurable neuromorphic structured AER systems. Application to convnets. *IEEE Transactions on Biomedical Circuits and Systems, 7*(1), 82–102.

Salvador Canas-Moreno is a researcher at department of Architecture and Tech. of Computers at the University of Seville. He received the B.S. degree in computer engineering and the M.S. degree in microelectronics from the University of Seville, Spain, in 2015 and 2016, respectively. His research interest includes VLSI digital design, embedded systems, neuromorphic computing for sensors, robotics, and deep-learning.



Enrique Piñero-Fuentes is a Ph.D. student at the department of Architecture and Tech. of Computers, member of the Robotics and Computer Technology Lab and possessor of a “Ayudas para la formación de profesorado universitario (FPU)” grant. He has a degree in Computer Engineering, having the best overall mark of the promotion. Research interest include ArtificiaI Intelligence, Machine Learning, Microcontrollers, FPGA, Edge-Computing, Robotics and Computer Vision.

Antonio Rios-Navarro received the B.S. degree in computer science engineering, the M.S. degree in computer engineering, and the Ph.D. degree in neuromorphic engineering from the University of Seville, Seville, Spain, in 2010, 2011, and 2017, respectively. He currently holds an Assistant Professor position at the Computer Architecture and Technology Department, University of Seville. His current research interests include neuromorphic systems, real-time spikes signal processing, field-programmable gate array design, deep learning and neural control.

Daniel Cascado-Caballero work as Assistant Professor in the University of Seville. He received his degree in Computer Science in 1996 and his PhD. in 2003, both from the University of Seville. From 2009 he has worked as Publication Chair of several International Conferences. His research field is focused on wireless communications, membrane computing, simulation and e-health systems, where he has numerous papers published in referred international journals and conferences.

Fernando Perez-Peña Fernando Perez Peña received the degree in telecommunication engineering from the University of Seville, Spain, in 2009, and the Ph.D. degree (specialized in neuromorphic motor control) from the University of Cadiz, Spain, in 2014. In 2015, he was a Postdoctoral Researcher with CITEC, Bielefeld University, Germany. He has been an Assistant Professor with the University of Cadiz, since 2014. His current research interests include neuromorphic engineering, FPGA digital design, motor control, and neurorobotics.

Alejandro Linares-Barranco is full-professor at the department of Architecture and Tech. of Computers and chair of the Neuromorphic Systems group of the Excelence Unit SCORE at the University of Seville. He is member of the Robotics and Computer Technology Lab since 2001. He received the B.S. degree in computer engineering, the M.S. degree in industrial computer science, and the Ph.D. degree in computer science from the University of Seville, Spain, in 1998, 2002, and 2003, respectively. His research interest includes VLSI digital design, embedded systems, neuromorphic computing for sensors, robotics, and deeplearning.