# *Neuro-inspired system for real-time vision sensor tilt correction*

A. Jimenez-Fernandez, J.L. Fuentes-del-Bosh, R. Paz-Vicente, A. Linares-Barranco, G. Jiménez
Dpto. Arquitectura y Tecnología de Computadores. Universidad de Sevilla. Av. Reina Mercedes s/n, 41012-Sevilla, SPAIN
ajimenez@atc.us.es

**Abstract— Neuromorphic engineering tries to mimic biological information processing. Address-Event-Representation (AER) is an asynchronous protocol for transferring the information of spiking neuro-inspired systems. Currently AER systems are able sense visual and auditory stimulus, to process information, to learn, to control robots, etc. In this paper we present an AER based layer able to correct in real time the tilt of an AER vision sensor, using a high speed algorithmic mapping layer. A co-design platform (the AER-Robot platform), with a Xilinx Spartan 3 FPGA and an 8051 USB microcontroller, has been used to implement the system. Testing it with the help of the USBAERmini2 board and the jAER software.**

## I. INTRODUCTION

In this paper we face a common problem in robot related to vision system. Some robots don't have usually a horizontal vision (like mobile robots, unmanned aerial vehicles), so vision systems need to correct its tilt to ensure a correct image processing. Thinking in ourselves, if we look to a horizontal line, and tilt horizontally our head, we don't lose the horizontal references, watching the line always horizontal. That is thanks to the sense of balance, which sensors can be found in the inner ear. In this paper we present a hardware layer able to correct a neuro-inspired vision sensor [1] information tilt in real time, taking advantage of how this kind of sensor represents visual information.

Bio-Inspired and Neuro-Inspired systems or circuits are approaches to solve real problems by mimicking the biology in its efficient solutions. Spiking systems is one of the neuro-inspired alternatives of mimicking the neurons layers of the brain for processing purposes. Vision spiking systems process the information into a continuous way, without discretization of the visual information into frames. Hardware implementations of these spiking systems are usually composed by several steps: sensors [1][2], filters [3], convolutions [4][5][6], actuators[11], etc… Each of these steps consists in one or several chips that has to process the information like in the brain: they establishes point to point connections between neurons from one layer or chip to other or other neurons of the next layer or chip. Engineers found a great problem at this point because they need to communicate thousands of neurons from one chip to the next chip, but they have a limitation in the number of pins. Address-Event-Representation solves this problem [7].

AER was proposed by the Mead lab in 1991 [7] for communicating between neuromorphic chips with spikes (Fig. 1). Each time a cell on a sender device generates a spike, it communicates with the array periphery and a digital word representing a code or address for that pixel is placed on the external inter-chip digital bus (the AER bus). Additional handshaking lines (Acknowledge and Request) are used for completing the asynchronous communication. In the receiver chip the spikes are directed to the pixels whose code or address was on the bus. In this way, cells with the same address in the emitter and receiver chips are virtually connected by streams of spikes. Cells that are more active access the bus more frequently than those less active. Arbitration circuits usually ensure that cells do not simultaneously access the bus. Usually these AER circuits are built using self-timed asynchronous logic [8].

There is a growing community of AER developers for bio-inspired applications in vision, audition systems, robot control, etc. As demonstrated by the success in the last years of the AER group at the Neuromorphic Engineering Workshop series [9], and most recently the Capo Caccia Cognitive Neuromorphic Engineering Workshop [10]. The goal of this community is to build large multichip and multi-layer hierarchically structured systems capable of performing massively-parallel data-driven processing in real time. The success of these systems will strongly depend on the availability of robust and efficient development, debugging and interfacing AER-tools [15].

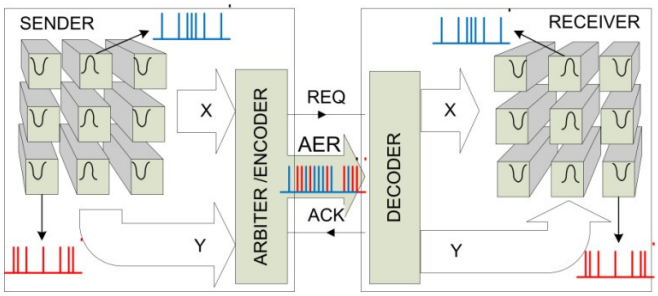


Figure 1. Rate-Coded AER inter-chip communication scheme.

In the following sections we present and describe the AER system developed. In section II we focus on the problems of vision sensors regarding to tilt variations and we propose

mechanism for correcting them. Section III presents full system implementation. Finally section IV describes test scenario and experimental results.

## II. AER PROCESSING ARCHITECTURES

AER vision sensors output is composed by a stream of a parallel number of bits, called AER events, where each event address represent a pixel coordinate (x, y). Its frequency is a function of pixel activity. One example is the Dynamic Vision Sensor (DVS128) [1] designed by the Institute of Neuro Informatics (INI) at Zurich. DVS128 provides an AER output stream where each event frequency is proportional to the derivate of the luminosity in time. So for these feature it needs to add event polarity (p), because events can be positive (pixel luminosity changing from brighter to darker), or negative otherwise. DVS128 have a resolution of 128x128 pixels, and its AER output event is codified as showed in Figure 2.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| N.C. | Y | | | | | | | X | | | | | | | P |

Figure 2. DVS128 output events structure

One advantage of transmitting the pixels addresses is that we can perform extra operations on the events while they travel from one chip to another. For example the output of the DVS128 can be translated, scaled, or rotated by mapping operations on the emitted addresses, only modifying them with the desired operation. Unlike traditional vision system, those need to store a complete digital video frame, to process information pixel by pixel, and after that send processed frame sequentially.

When this sensor is used in mobile robots applications (wheeled or planes), the sensor suffers variations due to the gravity. In that case the output provided is representing not only changes in visual field, but else these changes due to changes in sensor position or vibrations.

For DVS128 tilt correction, we need to focus our attention in image rotation operation. As is well known, we can rotate a 2-D image around a coordinate according next rotation matrix:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} \left( \begin{pmatrix} x \\ y \end{pmatrix} - \begin{pmatrix} x_c \\ y_c \end{pmatrix} \right)$$

Where x and y are original coordinates, $x_c$ and $y_c$ represents the rotation center, x' and y' the rotated coordinates, and α is the rotation angle.

Our tilt correction layer must receive events from retina (original coordinates, x and y) and retina tilts (rotation angle, α). The retinal tilt can be measured by commercial sensors like accelerometers or gyroscopes. Then this layer has to process event coordinates according to rotation matrix, and generate a new event address tilt corrected (rotated coordinates, x' and y'). Finally the mapped events must represent the rotation angle applied to the input event. This process can be implemented by simple mapping operations [12] based on look-up tables (LUT) or algorithmic mappings.

Using a LUT based solution would need important quantities of memory (e.g. 8Mbytes for the DVS128 with 128 different tilt degrees), and several system clock cycles for memory access. To avoid these problems, leveraging the power of modern programmable logic devices (like FPGA), we can perform a parallel algorithmic event processing. This means that we are not going to have in a memory every rotation coordinates. We need to calculate in real time each output event address every time an event arrives, according to the tilt degree. For this task we are going to design a digital circuit (written in VHLD) that perform this operation, applying the rotation matrix to incoming events coordinates in real-time and parallelizing every operation. Next section explains this mechanism implementation.

## III. AER TILT CORRECTION LAYER

In this section we present the AER Tilt Correction Layer (ATCL). We have implemented the ATCL for the AER-Robot platform [11], which can be seen at the bottom of Figure 6. The AER-Robot platform was originally designed to control robots, as a bridge between AER buses and robots actuators, however it is enough versatile and powerful to be used as vision processor. This platform is a co-design platform with FPGA and microcontroller architecture. It includes a Xilinx Spartan 3 FPGA (XC3S400) and an 8051 microcontroller (C8051F320). FPGA allows high speed AER event processing, while the microcontroller is slower, but has analog to digital converters (ADC) and USB port, extending the capabilities of the FPGA.

This layer has been conceptually divided in two different main blocks, one for measure DVS128 tilt, and other for process the AER visual information according to the DVS128 tilt. We have distributed these blocks along the AER-Robot microcontroller and FPGA. Showed in Figure 3. where in top is the tilt measuring block, and in bottom the AER events processing block.

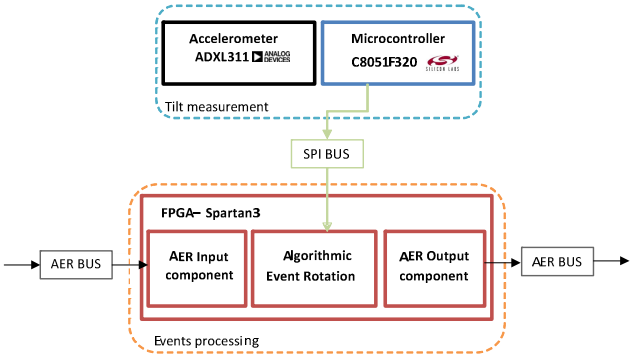


Figure 3. AER tilt correction layer main blocks

### A. *Titl meassuring block*

The block for tilt measuring is located inside the 8051 microcontroller, who measures the DVS128 tilt from an accelerometer, and sends tilt information to the FPGA using the Serial Peripheral Interface (SPI) [15] for communication. It can be seen if the top of Figure 3.

Tilt is measured from an analog accelerometer (ADXL311). These kinds of sensors are able to measure gravity acceleration, being their output proportional to sensor tilt. ADXL311 output is connected to one of the 8051 microcontroller ADC inputs, converting it to a 8 bits digital

value. For image rotation we need the sine and cosine value of the tilt angle, so instead to transfer the tilt angle to the FPGA, we have stored inside the microcontroller a LUT with sine and cosine values for each angle, sending to the FPGA directly these values. This solution is faster because sine and cosine calculation can heavily load the FPGA, and because it's enough a precision of 128 different tilt angles, so it only needed 256 bytes of 8051 memory. We can see how microcontroller stores sine and cosine LUT in Figure 4, Figure 4. where tilt angle is in the top, and LUT positions in the bottom. To avoid using floating point operations, sine and cosine has been stored scaled by 128; it means that these values have a range between ±127. Every time an ADC conversion is done, we take sine and cosine values from conversion value memory position, and finally are sent though the SPI port to the FPGA.

| Tilt Angle | ≈-90º | ≈0º | ≈90º |
|---|---|---|---|
| Cosine | 0 | 127 | 0 |
| Sine | -127 | 0 | 127 |
| Memory Address | 0 | 63 | 127 |

Figure 4. Sine and Cosine lookup table inside the microcontroller.

*B. AER events processing block*

This block has to be able to receive AER events, to compute the rotation, and to send new events. The AER events processing block internal components are showed in the bottom of Figure 3. At left input events are received by a component that implements the AER protocol for input events. Once an event is received, it coordinates are transferred to the algorithmic event rotation block. Finally, rotated event is transmitted to the next layer using an AER output component, at right.

Algorithmic event rotation block has two inputs, the incoming event coordinates, and rotation information (from the SPI port). Rotation is received by an SPI slave entity, which manage SPI communication and receives sine and cosine values from the microcontroller. A second element is performing the AER event coordinates rotation by accessing to two registers with the rotation information.

At this point we have the information of an event coordinates, x and y, and the tilt angle sine and cosine value. The only remaining operation is to apply the rotation matrix to get a new rotated event. For that we use asynchronous adders and multipliers embedded inside the FPGA, performing this computation in a single clock cycle. The circuit that implements the rotation matrix can be seen in Figure 5. Input values are on the top: event coordinates, x and y, rotation center coordinates, fixed to 64, and tilt angle sine and cosine values obtained from microcontroller LUT. A sequence of adders and multipliers apply the rotation matrix to the incoming AER event. Both AER event coordinates has 7 bits, and tilt angle sine and cosine values have 8 bits, in consequence when we multiply them we get a 15 bits value. Remembering that tilt angle sine and cosine values have been scaled by 128, we need to divide multiplication results by 128, ignoring 7 less significant bits, which is the new 7 bit output coordinate. Finally, coordinates x' and y' are joined in a new AER event, that is transmitted through the AER output port.

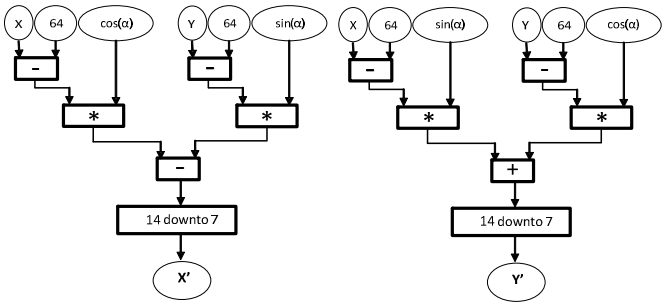


Figure 5. Block diagram of AER rotation entity.

VHDL simulation shows that AER event rotation takes 5 clock cycles, 2 for AER input communication, only 1 to event rotation within, and last 2 clock cycles for AER output event transmission. VHDL synthesis report denotes that our implementation needs 498 slices (from 3584 that haves our Spartan 3, about 13%), and can operate at 80.14MHz.

AER event processing block performs 12 arithmetic operations per event (5 subtractions, 1 addition, 4 multiplications, and 2 divisions). With an 80MHz clocked system we can reach an ideal maximum of 960 Mops (Mega operations per second). However AER communication takes 4 clock cycles, decreasing our system performance to 192 Mops, being the AER communication the bottleneck of our system. ATCL only introduces a latency of 62.5nS, able to process 16M events per second.

## IV. SYSTEM TEST SCENARIO

For testing purposes we need to excite ATCL with an AER video streaming, and to monitor ATCL output, while moving the accelerometer. For this we use the USBAERmini2 board [13] as hardware interface, and the jAER software [14] as host interface. Complete system test scenario is shown in Figure 6. Where the USBAERmini2 is on the top, connected to the AER-Robot by two AER ports, on the bottom (one for AER input and other for output), and the accelerometer, that should be attached to a robot, in the middle.

The USBAERmini2 is a bridge between an USB port and AER buses. It is able to monitor AER traffic trough the USB port, and also to sequence AER information. Its features can be found detailed in [13]. The idea is to sequence AER information from a PC using its AER sequencer port, process this information inside the ATCL according to accelerometer tilt, and monitor ATCL response using the AER monitor port.

In the PC we use the jAER software [14], it is Java open source software that manages the USBAERmini2. Allowing us to select sequencing AER files comfortably, and visualize ATCL rotated video streaming in the PC screen.

Experiments results are shown in Figure 7. There we can see two screens captures from jAER, images are from an AER sample file from jAER web site, on figure top we can see a person juggling but rotated about 30º to the left, and on the

bottom, the same image rotate to the right. This happens according to accelerometer tilt, correcting image tilt in real-time and in a continuous way.

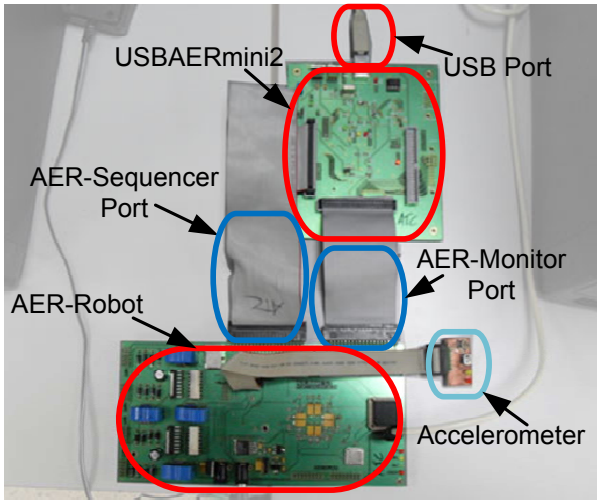


Figure 6. ATCL test scenario

## V. CONCLUSIONS

This paper presents an AER layer for tilt correction, the ATCL. ATCL is software / hardware co-designed system, based on an 8051 microcontroller and a Xilinx FPGA, describing complete system deeply in this paper. Finally a test scenario has been designed and results showed, thanks to the use of AER-tools from the AER community. ATCL faces a common problem in several robotic fields, and can be easily integrated to current robots with AER vision.

## ACKNOWLEDGMENTS

This work has been supported in part by the Andalucía Council with the BrainSystem project (P06-TIC-01417), and by the Spanish projects: SAMANTA II (TEC2006-11730-C03-02) and VULCANO (TEC2009-10639-C04-02)

## REFERENCES

[1] P. Lichtsteiner, et al. "A 128×128 120dB 15 us Asynchronous Temporal Contrast Vision Sensor". IEEE Journal on Solid-State Circuits, 2008.

[2] V. Chan, et al. "AER EAR: A Matched Silicon Cochlea Pair with Address-Event-Representation Interface". IEEE International Symposium on Circuits and Systems, 2007. ISCAS 2007.

[3] R. Serrano-Gotarredona, et al. "AER Building Blocks for Multi-Layer Multi-Chip Neuromorphic Vision Systems". NIPS 2005.

[4] R. Serrano-Gotarredona, et al.. "On Real-Time AER 2-D Convolutions Hardware for Neuromorphic Spike-Based Cortical Processing. IEEE Transactions on Neural Networks, Vol. 19, No 7, pp. 1196-1219. July-2008.

[5] Oster, M et al "Quantifying Input and Output Spike Statistics of a Winner-Take-All Network in a Vision System" IEEE International Symposium on Circuits and Systems, 2007. ISCAS 2007.

[6] P. Hafliger. "Adaptive WTA with an Analog VLSI Neuromorphic Learning Chip". IEEE Transactions on Neural Networks, vol. 18, No 2,. March-2007.

[7] M. Sivilotti, Wiring Considerations in analog VLSI Systems with Application to Field-Programmable Networks, Ph.D. Thesis, California Institute of Technology, Pasadena CA, 1991.

[8] Kwabena A. Boahen. "Communicating Neuronal Ensembles between Neuromorphic Chips". Neuromorphic Systems. Kluwer Academic Publishers, Boston 1998.

[9] Telluride Neuromorphic Engeeniering Workshop 2008: https://neuromorphs.net/ws2008/wiki/

[10] The 2009 Capo Caccia Cognitive Neuromorphic Engineering Workshop: http://capocaccia.ethz.ch/capo/

[11] A. Linares-Barranco et al. "AER Neuro-Inspired interface to Anthropomorphic Robotic Hand". IEEE World Conference on Computational Intelligence. IJCNN. Vancouver, July-2006.

[12] A. Linares-Barranco et al. "Implementation of a time-warping AER mapper". IEEE International Symposium on Circuits and Systems. ISCAS 2009.

[13] R. Berner, et al. "A 5 Meps $100 USB2.0 Address-Event Monitor-Sequencer Interface". IEEE International Symposium on Circuits and Systems ISCAS 2007.

[14] jAER open-source software project. http://jaer.wiki.sourceforge.net/

[15] R. Serrano-Gotarredona, et al. "CAVIAR: A 45k-neuron, 5M-synapse AER Hardware Sensory-Processing-Learning-Actuating System for High-Speed Visual Object Recognition and Tracking," IEEE Trans. on Neural Networks, Volume 20, Issue 9, Sept. 2009.

[16] J. Catsoulis. "Designing Embedded Hardware". O'Reilly. ISBN: 0-596-00755-8.

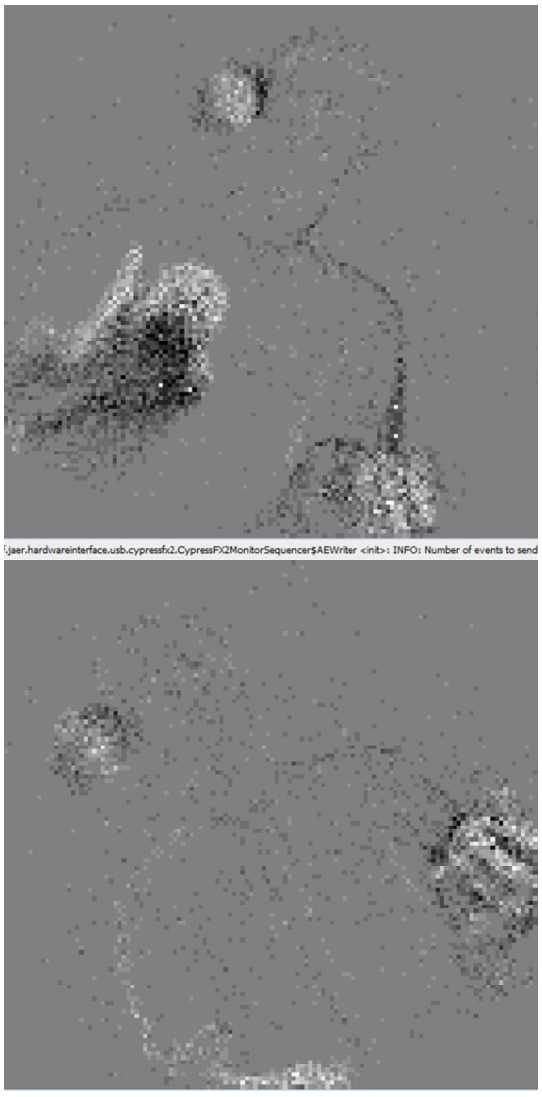


Figure 7. jAER captures for two different tilt angles