Conradt J, Cook, M., Berner, R., Delbruck T:
An Embedded AER Dynamic Vision Sensor for Low-Latency Pole Balancing.
In 5th IEEE Workshop on Embedded Computer Vision (in conjunction with ICCV 2009); Kyoto, Japan: IEEE: 2009:1-6.

# An Embedded AER Dynamic Vision Sensor for Low-Latency Pole Balancing

Jorg Conradt, Raphael Berner, Matthew Cook, Tobi Delbruck
Institute of Neuroinformatics, UZH and ETH-Zürich
Winterthurerstr. 190, CH-8057 Zürich
{conradt, raphael, cook, tobi}@ini.uzh.ch
http://www.ini.uzh.ch/~conradt/projects/PencilBalancer

## Abstract

*Balancing small objects such as a normal pencil on its tip requires rapid feedback control with latencies on the order of milliseconds. Here we describe how a pair of spike-based silicon retina dynamic vision sensors (DVS) is used to provide fast visual feedback for controlling an actuated table to balance an ordinary pencil on its tip. Two DVSs view the pencil from right angles. Movements of the pencil cause spike address-events (AEs) to be emitted from the DVSs. These AEs are processed by a 32-bit fixed-point ARM7 microcontroller (64MHz, 200mW) on the back side of each embedded DVS board (eDVS). Each eDVS updates its estimate of the pencil's location and angle in 2d space for each received spike (typically at a rate of 100kHz) by applying a continuous tracking method based on spike-driven fitting to a model of the vertical rod-like shape of the pencil. Every 2ms, each eDVS sends the pencil's tracked position to a third ARM7-based controller, which computes pencil location in 3d space and runs a linear PD-controller to adjust X-Y-position and velocity of the table to maintain the pencil balanced upright. The actuated table is built using ordinary high-speed hobby servos. Our system can balance any small, thin object such as a pencil, pen, chop-stick, or rod for minutes, in a wide range of light conditions.*

## 1. Introduction

Balancing an object has been used for many years as a demonstration of controller design. Such demonstrations in teaching robotics and robotics contests often are limited to a single pole rotating about one constrained axis for simplicity, and typically use a position encoder at the bottom of the object providing the current angle relative to desired balanced orientation, as opposed to using purely visual input.

Normal image sensors are hard to use for balancing small objects because the frame rate limits the response latency, necessitating complex nonlinear control methods that can control despite the large signal nonlinearities that develop with controller delay. One example available online [1] shows a Sarcos industrial robot balancing a pole approximately 1m long with a weighted top and two colored markers which are used for tracking. Since the angular acceleration by gravity is inversely proportional to the distance of the center of mass (COM) from the base, the weighted top shifts the center of mass away from the hand, easing the task significantly. An unmodified rod of length L has its COM $c = \frac{1}{2} \cdot L$ from the base. The angular acceleration ($\ddot{\alpha}$ in rad/s$^2$) caused by gravity (g) for small angle α (where $sin(\alpha) \approx \alpha$) is given by $\ddot{\alpha} = (\alpha \cdot g)/c$, representing an angle that increases by a factor of e every sqrt(c/g) seconds. For a normal pencil of length L=20cm, the angle increases 10% every 10ms. A simple linear controller must react within a few ms to balance such an object.

Here we show how the use of data-driven spike based vision sensors with embedded microcontrollers facilitates the application of straightforward linear control policies with low computational requirements, that allows visually guided balancing of normal pencils, which otherwise represents a challenging control problem. The paper describes the vision sensors and their spiking output data, the embedded hardware (64MHz 32bit microcontroller) to process such spiking events in real-time, an algorithm to compute a pencil's position in 3d space, the linear control policy, the balancing robot, and issues related to the embedded fixed-point implementation. We present and discuss measurements from the running system, and conclude with a comparison using conventional image sensors. This work improves on [4] which described a non-embedded version of the balancer which ran similar algorithms on a host PC and used USB interfaces to the DVS sensors and servo controller. In the present work, CPU power consumption has been reduced by a factor of about 100 and performance has been substantially improved by the use of dedicated real-time hardware, yielding the first stand-alone vision-based balancing system for pencil-sized objects.

## 2. Robotic Hardware

This section describes the vision sensor, embedded processing hardware, and robotic setup.

### 2.1. Dynamic Vision Sensor

The dynamic vision sensor (DVS) [2][3] used as a pair in this project is an address-event silicon retina that responds to temporal contrast (Fig 1). Each output spike address represents a quantized change of log intensity at a particular pixel since the last event from that pixel. The address includes a sign bit to distinguish positive from negative changes. All 128x128 pixels operate asynchronously and signal illumination changes within a few microseconds after occurrence. No frames of "complete" images exist in the system, but instead only individual events that signal changes at a particular spatial position denoted by a particular pixel's address.

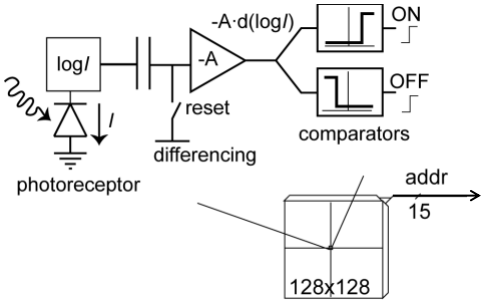


Figure 1: DVS pixel and camera architecture. The DVS pixel outputs events that represent quantized log intensity changes as shown by the simplified pixel schematic (top left). After transmission of the pixel address (addr), the pixel is reset.

The standard DVS contains an on-board digital logic chip with high-speed USB 2.0 interface (not shown in Fig 1) that takes addresses and delivers time-stamped addressevents with a resolution of 1us to a host PC for off-line processing. The DVS specifications are summarized in table I.

TABLE I. THE DVS'S ELECTRICAL SPECIFICATIONS

| | |
|---|---|
| Dynamic range | 120 dB |
| Contrast threshold mismatch | 2.1% |
| Pixel array size | 128x128 pixels of $(40u)^2$ |
| Photoreceptor bandwidth | >= 3 kHz |
| Event saturation rate | 1 M-event per second |
| Power consumption | 23 mW |

### 2.2. The Embedded DVS Board

For small embedded DVS applications, such as found in mobile robotics, a USB communication channel to report raw events is not necessary. We developed a small embedded DVS system (eDVS, Fig 2) composed of a DVS chip directly connected to a 64MHz 32bit microcontroller (NXP LPC2106/01) with 256kbyte on-board program flash memory and 64kbyte on-board RAM. This processor initializes the DVS chip and captures events for immediate processing, following a simple handshaking protocol: Each occurring event from the DVS is transmitted as a 15-bit address (7 bits x-position, 7 bits y-position and 1 bit polarity), together with a request signal. The LPC2106 reads the address and acknowledges reception of that event, after which the DVS removes the request signal.

The LPC2106/01 microcontroller offers several communication ports, of which we provide connections to a simple two-wire-interface (TWI, ≤400kbaud), a universal asynchronous receiver/transmitter (UART, ≤4Mbaud), and a four-wire serial peripheral interface (SPI, ≤32Mbaud). The TWI allows chaining a large number of eDVS boards, e.g. on a mobile robot that requires 360-deg field-of-view. The UART port facilitates connections to a PC for setup, debug output, and reprogramming. The SPI provides an option for high speed data transfer, or to record incoming events directly on a compact flash memory card. In this project we connect the SPI to a liquid crystal display (LCD) with a resolution of 128x128 pixels, to display the sensors' events and line-tracking information.

The eDVS contains separate on-board voltage regulators (1.8V and 2x3.3V) for the microcontroller and the DVS sensor to reduce sensor noise. A similar eDVS board optimized for small size and weight uses a single voltage regulator with only marginal performance penalty. The DVS board running at full 64MHz processing power draws less than 200mW, and can get easily powered from a single LiPo cell, or – if desired – over a USB connection. Reducing the LPC2106/01's system clock significantly reduces the eDVS's power consumption, down to about 50mW for power sensitive applications.

The eDVSs' lenses in this project are taken from a tiny off-the-shelf video camera with a diagonal field of view of 40°. The embedded board measures 52x23mm, with a height of 6mm (30mm with lens) at a weight of 5g (12g with lens).

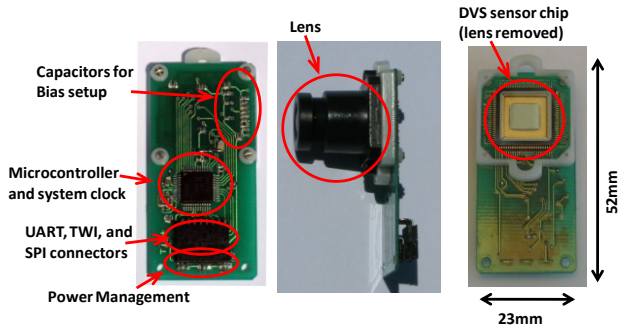


Figure 2: Embedded-DVS board showing functional elements.

After programming, the eDVS board can run applications that process the sensor events in real time. Sensor maintenance (such as adjusting bias values or acknowledging events) causes only marginal overhead. However, the microcontroller can adjust sensor settings during operation, e.g. to limit the event rate by dynamic control of the pixel contrast threshold. In this project the eDVS performs the line tracking task described in Sec. 3.1.

### 2.3. Balancer Hardware

The balancer hardware (Fig. 3) consists of a custom-built table capable of moving its hand cup within a range of 100x100mm. This hand cup is a 1cm radius conical depression formed from hard rubber which was machined while frozen with liquid nitrogen. A standard milling bit was used to machine the cup. The rubber provides sufficient friction to be able to rapidly accelerate the tip of the pencil without slippage, but does not otherwise support it.

The table is actuated by two independent servo motors. Both rotary heads of these servos connect via independent passive 2-segment arms of length 2x100mm to the hand cup. The angular positions of both servo-heads together define a unique position of the hand on the table. The lengths of both arms and the type of servos (Futaba BLS451, brushless coreless digital, maximum 500Hz pulse rate) have been selected to yield sufficient torque and fast response times, thus maximizing the speed of motion. The servos have internal feedback to autonomously reach a desired rotation angle. A 32-bit microcontroller (NXP LPC2106/01, same type as on each of the eDVS boards) running at 64 MHz computes PWM servos commands to reach a desired hand cup position in x-y-space on the table. The servos receive such updated control input at a frequency of 500 Hz, and measurements show that changes in PWM width cause movement of the hand in less than 12ms.

The microcontroller communicates with two independent eDVS boards, which each track the pencil to be balanced in 2d at right angles to each other (Fig 3). The microcontroller for servo control thus receives two independent 2d-tracking estimates, combines both into a 3d-position, and computes motion commands for balancing (refer to Sec. 3.1 for details). Additionally, the servo microcontroller can be interfaced to a host PC, allowing the PC to specify desired hand positions for testing purposes and to update the microcontroller's program.

The robot is powered either from a standard laptop power supply or from a LiPo battery pack. A LiPo pack of 7.2V and 6Ah can run the robot for several hours.

The cost of assembling the entire table was about US$1500 and was strongly dominated by the custom machining cost for the servo arms, which are aluminum that is milled out for low moment of inertia.

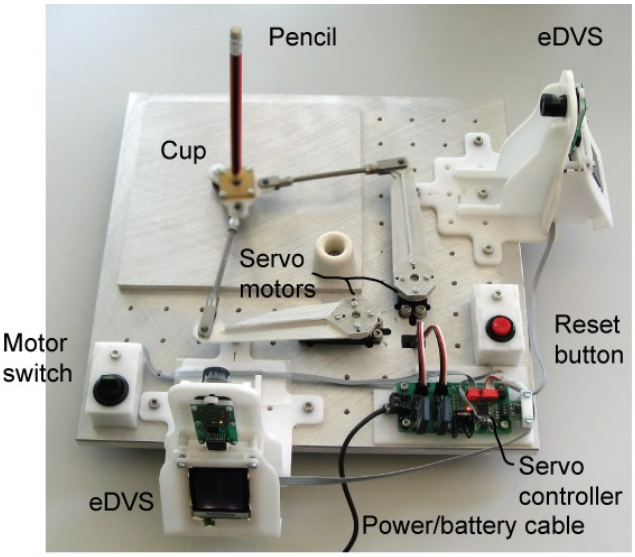


Figure 3: Photo of balancer hardware: 2 DVS (right top and bottom left), the motion table (top left) actuated by two servos (center)

## 3. Algorithms

This section describes the event-driven pencil tracking algorithm, the control system, and their implementation on the embedded processors.

### 3.1. Pencil Tracking Algorithm

The algorithm we use to track the pencil proceeds in two stages [4]. The first stage, done independently for each vision sensor on its eDVS processor, uses each incoming event to efficiently update an estimate of the line where the pencil appears to be from that vision sensor's point of view. The second stage, done on the servo controller, combines the two line estimates from the two vision sensors, taking perspective into account, to generate a 3d estimate of the line containing the pencil.

The goal of the first stage is to identify the line corresponding to where the pencil is in vision sensor coordinates from the sensor's point of view. Every event that arrives is treated individually to update the estimate of this line. The estimate of the line is maintained as a Gaussian in the "Hough space" of possible lines. Since the log of the Gaussian is a quadratic, our estimate of the line is stored as just the 5 coefficients of the quadratic (not including the constant term). This allows our estimate to be stored very compactly, with no discretization.

The equation for the line, $x=m \cdot y+b$, describes a line in m and b (the two dimensions of the Hough space) when x and y are given by the event. To update the quadratic for a newly arrived event, we simply decay the old quadratic slightly, replacing the decayed portion with the information from the new event. Since the new event corresponds to a line in the Hough space, it contributes a quadratic which

has its minimum on this line, and grows parabolically away from the line. This quadratic has coefficients $A=y^2$, $B=2\cdot y$, $C=1$, $D=-2\cdot x\cdot y$, and $E=-2\cdot x$, given an event from pixel (x,y). When we need to produce an estimate of the slope and x-intercept of the pencil, we report the lowest point of the quadratic, which is given by $(b,m) = (D\cdot B - 2\cdot A\cdot E, B\cdot E - 2\cdot C\cdot D)/q$, where $q=4\cdot A\cdot C-B\cdot B$.

The second stage combines the two lines from the first stage into a single line in 3d space. If the true position of the pencil is given by $(x,y,z)=(X+t\cdot\alpha_X, Y+t\cdot\alpha_Y, t)$, then the sensors at (0,-yr,0) and (xr,0,0) will see the line as:

$$(x,z) = ( (X+t\cdot\alpha_X)/(Y+t\cdot\alpha_Y+yr),\quad t/(Y+t\cdot\alpha_Y+yr) )$$
$$(y,z) = ( (Y+t\cdot\alpha_Y)/(xr-X-t\cdot\alpha_X),\ t/(xr-X-t\cdot\alpha_X) ).$$

We receive these as a base and slope for each sensor:

$$b1 = (X/(Y+yr), 0)\quad s1 = dx/dz = \alpha_X - X\cdot\alpha_Y/(Y+yr)$$
$$b2 = (Y/(xr-X), 0)\quad s2 = dy/dz = \alpha_Y + Y\cdot\alpha_X/(xr-X).$$

We can solve these for $X,\alpha_X,Y,\alpha_Y$ in terms of b1,s1,b2,s2:

$$\begin{aligned} X &= (b1\cdot yr + b1\cdot b2\cdot xr) / (b1\cdot b2+1) \\ \alpha_X &= (s1 + b1\cdot s2) / (b1\cdot b2+1) \\ Y &= (b2\cdot xr - b1\cdot b2\cdot yr) / (b1\cdot b2+1) \\ \alpha_Y &= (s2 - b2\cdot s1) / (b1\cdot b2+1) \end{aligned} \qquad [1]$$

This yields the position $(X,Y)$ of the pencil at the height of the cameras, as well as the slope of the pencil $(\alpha_X,\alpha_Y)$.

### 3.2. Control System

The pencil tracking algorithm reports its estimate of the pencil's position in 3d space, represented as a pair of position coordinates $(X,Y)$, and corresponding slopes in x and y directions $(\alpha_X,\alpha_Y)$. A PD-controller running in the microcontroller for servo control generates desired hand positions $(X_{des},Y_{des})$ based on these four inputs and the position time derivatives $(\dot{X},\dot{Y})$. In our system all final desired target values (positions, slopes, and velocities) are zero to keep the pencil upright in the center of the table. $X_{des},Y_{des}$ are computed as follows:

$$X_{des} = g_P X + g_\alpha \alpha_x + g_D \dot{X}$$
$$Y_{des} = g_P Y + g_\alpha \alpha_y + g_D \dot{Y}$$

Here $g_P$, $g_\alpha$, and $g_D$ denote the gain parameters for base position, slope, and base velocity, respectively. Selecting $g_P = 1$ (with $g_\alpha$, $g_D = 0$) moves the hand exactly underneath the current center of the pencil; whereas values of $g_p$ slightly larger than 1 move the hand further outside, helping the pencil to tilt backwards towards the center of the table. Intuitively, the middle term has a similar effect based on the pencil's current slope: for larger gains $g_\alpha$, the current tilt of the pencil more strongly influences the future position of the hand. The last term counteracts recent drift of the pencil.

We found a large range of gain settings for which the system exhibits consistent performance. Typical settings are:

$$g_P \approx 1.3\pm0.2 \quad g_\alpha \approx 250\pm100 \quad g_D \approx 70\pm20$$

### 3.3. Embedded Implementation

The algorithms described in the previous sections run on three fixed point 32-bit microcontrollers at relatively low clock frequencies of 64MHz. These controllers offer limited on-board memory (64Kbytes). They do provide a single-cycle fixed point hardware multiplier with 64-bit result, but no floating point hardware.

The motion control algorithm of Sec. 3.2 running on the servo controller computes an update of the pencil's 3d-position and new motor commands every 2ms in floating point; this update rate is slow enough to allow us to use floating point emulation.

By contrast, the pencil tracking eDVS microcontrollers receive new spike event addresses at rates of up to 200kHz and therefore need to process each new event in less than 5us. To keep the computation fast and efficient, all parameters of the above equations are maintained as integers that are left-shifted by the largest power of 2 such that they do not exceed the 32-bit limit imposed by the microcontroller. This scaling is statically determined for each variable independently to achieve the highest precision possible. All results are rescaled by right-shifting to their original base before being used for further computations.

The Sec. 3.1 line tracking algorithm runs in a loop that reacts on new events, and requires one division to compute a new estimate of current base $(X,Y)$ and slope $(\alpha_X,\alpha_Y)$. The formulas for these four values (Eqs. 1) all use the same denominator, so it is sufficient to compute the expression $den=1/(b1\cdot b2+1)$ once and do single cycle multiplications with $den$ for each of the four formulas.

The computation $1/(b1\cdot b2+1)$ cannot be performed within the available time window of <5 us. Hence, we only update the polynomial's coefficients A-E for every new event. The computational steps of the reciprocal are spread over up to 80 events. Hence the software only gets a new division result roughly every 80 events, which is sufficiently fast for new base and slope estimates to be requested by the motion controller every millisecond.

## 4. Results

Our system normally can balance an object for several minutes. We have not systematically investigated the causes for eventual failure because performance has been more than sufficient for demonstrations.

Events are processed in the eDVS microcontroller at up to around 500k events per second; the highest observed

event rate during operation is below 200k events/second for each sensor.

We used a variety of illumination sources ranging from uneven (shadowed) sunlight through the windows to fluorescent lighting of 300 lux to incandescent lighting from a table lamp. Low illumination degraded performance slightly by increasing DVS noise and decreasing pixel bandwidth, but generally the balancer is tolerant to a wide range of illumination conditions as long as it is relatively steady. We have demonstrated the balancer at several public exhibitions and during several seminars in a variety of lecture halls.

The following data is representative of the performance of the embedded balancer and is presented here for completeness.

Fig. 4 shows events emitted from one of the two eDVS due to pencil motion along the blue time axis. The solid black axes represent the field-of-view in sensor space: an event's position on the vertical axis corresponds to its height along the pencil; its position on the horizontal axis shows its displacement with respect to the center of the setup. Each black dot denotes a reported event at a given position in sensor space and time. The red pencil at t = 640ms shows the current estimate of pencil base and slope. This space-time plot shows oscillations of pencil position and tilt within the 640ms time window.

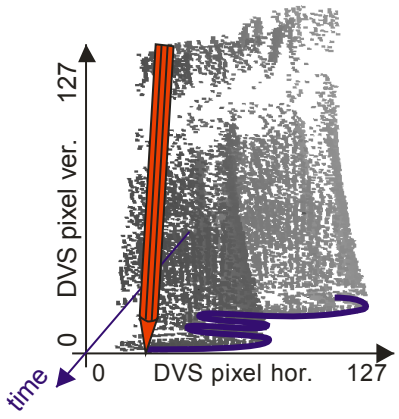


Figure 4: Space-time plot of 54k events (dots) reported from one eDVS sensor during balancing in a time window of 640ms. The pencil's base over time and the last tracked position are shown in blue and red. The lower density of events close to the top reflects lower contrast of the pencil's rubber holder in silver-metallic. From [4].

Fig. 5 shows recorded control data from our system during 2 seconds of operation. For clarity, we display data of a single dimension only. The top panel shows raw unfiltered data whereas the bottom panel shows the same data processed by a low-pass filter for clarity. The blue trace shows the estimated position of the pencil over time; the red trace a 100-fold amplification of the pencil's slope ($\alpha_X$). Both these signals are obtained based on spiking visual input only. The green trace shows the desired position of the cart, as computed in Sec. 3.2. The blue position trace follows the green desired position trace with an average delay of about 50 ms. This delay is probably dominant in limiting the possible object length that can be balanced.

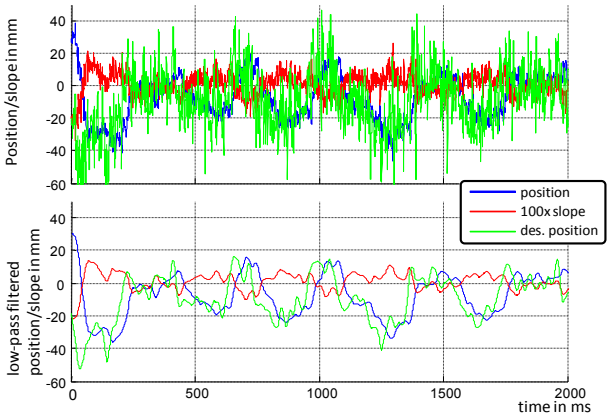


Figure 5: Recorded traces of position, slope and desired position during a 2s time window. Upper graph: raw data; lower graph: same data filtered in 3rd order Butterworth filter (-3dB cutoff frequency set to 30Hz) for clarity. From [4].

Fig. 6 is a histogram of X, Y-positions visited by the table during balancing. The plot clearly shows that the cart typically stays close to the center of the table, but occasionally needs much of the available motion space. The center of balancing is shifted relative to the table's origin, indicating an offset between the centers of the two DVS and the center of the table. In fact, we never properly calibrated the visual system with the actuated table.

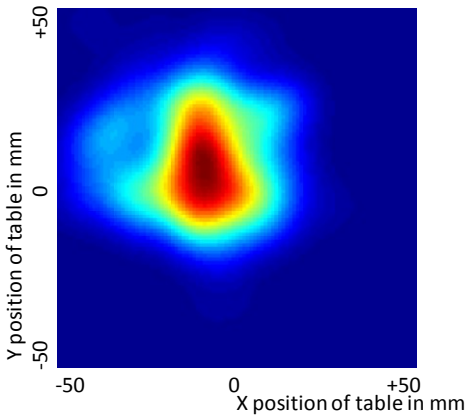


Figure 6: Histogram of relative occurrances of true table positions, red denoting areas often visited. From [4].

## 5. Conclusions

This paper describes a balancer demonstration that uses embedded spike-based vision sensors. The low latency and

sparse output of the sensors enable a straightforward solution to this balancing problem, which challenges conventional imaging based systems. A solution to balancing small objects using standard image sensors requires running at greater than 1kHz frame rate. Even at the relatively low spatial resolution of 128x128 = 16 k-pixels of the DVS sensors, image analysis requires processing pixels at a rate of 2·16k·1k=32M pixels/second for data acquisition, which would saturate a USB2.0 hub. Log conversion for temporal contrast extraction and subtraction against stored values to obtain event-like equivalents to the DVS output would require several hundred MIPs of processing before the remaining processing described here. Quantization noise at the low end of the conversion scale would limit low light performance severely, as would the 1ms exposure times, necessitating bright and uniform lighting. The use of embedded AER sensors and event-driven methods for computation has simplified and reduced the cost of implementing this demonstration, and has shown the advantages of the event-driven style of computation used in brains.

## References

[1] Available: http://www.youtube.com/watch?v=lwvTyC7m4LQ

[2] A 128×128 120dB 15us Latency Asynchronous Temporal Contrast Vision Sensor, (2007) Lichtsteiner, P., C. Posch and T. Delbruck. IEEE Journal of Solid State Circuits, Feb. 2008, 43(2) 566-576.

[3] Available: http://siliconretina.ini.uzh.ch

[4] J. Conradt, M. Cook, R. Berner, P. Lichtsteiner, RJ. Douglas, T. Delbruck, "A Pencil Balancing Robot using a Pair of AER Dynamic Vision Sensors" 2009 IEEE International Symposium on Circuits and Systems (ISCAS), 2009, 781-785.