frontiers in Robotics and AI

CODE
published: 16 January 2018
doi: 10.3389/frobt.2017.00073

The Event-Driven Software Library for YARP—With Algorithms and iCub Applications

Arren Glover, Valentina Vasco, Massimiliano Iacono and Chiara Bartolozzi**

iCub Facility, Istituto Italiano di Tecnologia, Genova, Italy

Event-driven (ED) cameras are an emerging technology that sample the visual signal based on changes in the signal magnitude, rather than at a fixed-rate over time. The change in paradigm results in a camera with a lower latency, that uses less power, has reduced bandwidth, and higher dynamic range. Such cameras offer many potential advantages for on-line, autonomous, robots; however, the sensor data do not directly integrate with current "image-based" frameworks and software libraries. The iCub robot uses Yet Another Robot Platform (YARP) as middleware to provide modular processing and connectivity to sensors and actuators. This paper introduces a library that incorporates an event-based framework into the YARP architecture, allowing event cameras to be used with the iCub (and other YARP-based) robots. We describe the philosophy and methods for structuring *events* to facilitate processing, while maintaining low-latency and real-time operation. We also describe several processing modules made available open-source, and three example demonstrations that can be run on the neuromorphic iCub.

Keywords: iCub, neuromorphic engineering, event-driven vision, software, humanoid robotics

OPEN ACCESS

Edited by:
Lorenzo Jamone,
Queen Mary University of London,
United Kingdom

Reviewed by:
Garrick Orchard,
National University of Singapore,
Singapore
Hanme Kim,
Imperial College London,
United Kingdom

****Correspondence:***
Arren Glover
arren.glover@iit.it;
Chiara Bartolozzi
chiara.bartolozzi@iit.it

Specialty section:
This article was submitted to Humanoid Robotics, a section of the journal Frontiers in Robotics and AI

Received: *26 July 2017*
Accepted: *12 December 2017*
Published: *16 January 2018*

Citation:
Glover A, Vasco V, Iacono M and Bartolozzi C (2018) The Event-Driven Software Library for YARP—With Algorithms and iCub Applications. Front. Robot. AI 4:73. doi: 10.3389/frobt.2017.00073

1. INTRODUCTION

Conventional vision sensors used in robotics rely on the acquisition of sequences of static images at fixed temporal intervals. Such a sensor provides the most information when the temporal dynamics of the scene match the sample-rate. If the dynamics are slower (e.g., a mostly static scene), only a small percentage of pixels change between two consecutive frames, leading to redundant acquisition and processing. Alternatively, if the scene dynamics are much faster (e.g., a falling object), information between images can be distorted by motion blur, or missed entirely.

A newly emerging technology, "event-driven" (ED) cameras, are vision sensors that produce digital "events" only when the amount of light falling on a pixel changes. The result is that the cameras detect only contrast changes (Lichtsteiner et al., 2008) that occur due to the relative motion between the environment and the sensor. There is no fixed sampling rate over time, instead, the sensor adapts to the scene dynamics. Redundant data are simply not produced in slow dynamic scenes, and the sensor output still manages to finely trace the movement of any fast stimuli. Specifically, the camera hardware latency is only 15 μs (Lichtsteiner et al., 2008) and the temporal resolution at which an event can be timestamped is under 1 μs. Events are also produced asynchronously for each pixel, such that processing operations can start without the need to read the entire sensor array, and a low-latency processing pipeline can be realized.

ED cameras provide many potential advantages for robotics applications. The removal of redundant processing can give mobile robots longer operating times and frees computational resources for other tasks. Fast-moving stimuli can always be detected, and visual dynamics estimated with more accuracy than with conventionally available cameras. This low-latency can enable extremely fast

reaction times between environmental change and the response of the robot. In addition, each pixel has a high dynamic range (143 dB (Posch et al., 2011)) which allows robots to operate in both bright and dark environments, and in conditions with widely varying intra-scene lighting. The sensor is low-power, promoting longer operation times for untethered mobile robots.

The *neuromorphic iCub* (Bartolozzi et al., 2011) is a humanoid robot that has a vision system comprised of two event cameras. The iCub robot is supported, in software, by the Yet Another Robot Platform (YARP) middleware (Metta et al., 2006), upon which the iCub low-level and application-level modules have matured using standard cameras, and also utilized other freely available algorithms (e.g., using OpenCV). However, due to the asynchronous nature of the event-stream, and its fundamental differences from 2D frame sequences (see **Figure 1**), traditional computer vision algorithms and image processing frameworks cannot be directly applied.

This paper introduces the event-driven software libraries and infrastructure that is built upon YARP and integrates with the iCub robot. The library takes advantage of the YARP framework, which enables the distributed processing of events within multiple interchangeable modules spread across multiple networked machines. Modules include pre-processing utilities, visualization, low-level event-driven vision processing algorithms (e.g., corner detection), and robot behavior applications. These modules can be run and used by anyone for purely vision-based tasks, without the need for an iCub robot by using: pre-recorded datasets, a "stand-alone" camera with a compatible FPGA, a "stand-alone" camera with the compatible USB connection, or by contributing a custom camera interface to the open-source library. As the processing is modular, the exact method of event acquisition is transparent to the remainder of the library. This paper also describes several iCub applications that have been built upon the ED cameras and library and highlights some recent experimental results. We begin with a brief description of the current state-of-the-art in ED vision for robotics.

2. EVENT-DRIVEN VISION FOR ROBOTS

Recent work using event cameras show promising results for fast, low-latency robotic vision. The latency of an event-based visual attention was two order less than frame-based one (Rea et al., 2013). Recognition of playing-card suit was achieved as a deck was flicked through (30 ms exposure) (Serrano-Gotarredona and Linares-Barranco, 2015). Detection of a moving ball by a moving robot was achieved at rates of over 500 Hz (Glover and Bartolozzi, 2016). Visual tracking of features was shown at a rate higher than standard cameras (Vasco et al., 2016a) and also features position could be updated "between frames" of a standard camera (Kueng et al., 2016).

The extreme low-latency of event cameras enabled fast close-loop control (e.g., inverse pendulum balancing (Conradt et al., 2009) and goal keeping with 3 ms reaction time and only 4% CPU utilization (Delbruck and Lang, 2013)). High-frequency visual feedback (>1 kHz) enabled stable manipulator control at micrometer scale (Ni et al., 2012). On-board pose estimation during flips and rolls of a quadrotor has been shown to be plausible using event-driven vision (Mueggler et al., 2015). Finally, robotic navigation and mapping systems include a real-time 2-DOF SLAM system for a mobile robot (Hoffmann et al., 2013), and 6-DOF parallel tracking and mapping algorithms (Kim et al., 2016; Rebecq et al., 2016).

Some of the above experiments used the Java-based jAER (Delbruck, 2008); however, Java is typically less suited to on-line robotics due to computational overheads. jAER is also designed to process events from a camera directly connected to a single machine; however, robotics platforms have come to rely on a middleware that distributes processing over a computer network. A middleware allows the modular connection of sensors, algorithms and controls, which are shared within the robotics community to more quickly advance the state-of-the-art. Perhaps the most well known is the Robot Operating System (ROS), in which some support for event cameras has been made available.[1] In this paper, we present the open-source libraries for event camera integration with the YARP middleware that is used on iCub.

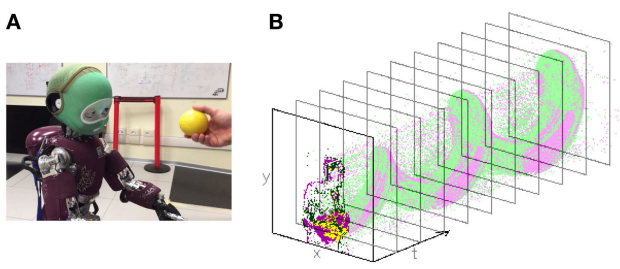

FIGURE 1 | The **(A)** iCub robot performing ball tracking and gazing toward the ball position and **(B)** the corresponding stream of events over time superimposed with approximate frame-captures for a hypothetical 10 Hz frame-based camera.

3. THE EVENT-DRIVEN LIBRARY

ED cameras encode information as a stream of asynchronous events with sub-μs resolution. When a pixel detects an illumination change beyond a threshold, it emits a digital pulse that can be assigned a timestamp and pixel address (using Address Event Representation (AER) (Mortara, 1998)) by a clock-based digital interface (e.g., FPGA or microcontroller). The entire visual information is, therefore, encoded within the relative timing and pixel position between events. An example event-stream is shown in **Figure 1**.

This event-driven library is designed to read events from the cameras, interface to communications for distributed processing, and provide event-based visual processing algorithms toward low-latency robotic vision. The library is written in C++, uses the `ev` namespace, and is integrated with the YARP middleware.

3.1. Representing an Event

The basic element representing an event is a `ev::vEvent`, which only stores the timestamp, i.e., *when* an event occurred. The

[1] github.com/uzh-rpg/rpg_dvs_ros.

information about *what* occurred is instead stored in the member variables of classes that are inherited from a `ev::vEvent`, see **Figure 2**. Events produced by the event cameras are called `ev::AddressEvent`, which consist of pixel location (x,y) and pixel polarity (p: darker/lighter) in addition to the camera channel (c: left/right). Algorithmic processing of events can be used to append additional information to an event, such as adding the velocity from an optical-flow algorithm. Currently used additional event-types include optical-flow events (`ev::FlowEvent`), class-labeled events (`ev::LabelledAE`), and events with a spatial distribution (`ev::GaussianAE`).

An instantiated `ev::vEvent` is wrapped in a C++11 `shared_ptr` such that memory is automatically managed, and events can be referenced in multiple threaded environments without duplicated memory allocation. The event-driven library provides a set of wrapper functions to ensure the `shared_ptrs` are correctly handled (see **Listing 1**).

These event-types can be easily extended through inheritance, and by defining the required additional member variables. Packet encoding and decoding methods are also required for transmission (described below). The framework is designed to be fully future compatible with the integration of different event-driven sensors (e.g., tactile and audio) by extending the base `ev::vEvent` class.

3.2. Event-Packets in YARP

On the iCub robot, a Linux driver reads the events from the camera FPGA interface and the `zynqGrabber` module exposes the data on a YARP port. A packet of events is sent in a `ev::vBottle` (a specialized type of `yarp::os::Bottle`) such that the bit-coding of the AER is preserved: to retain data-compression and compatibility with other AER-based hardware. A module that receives a `ev::vBottle` can decode the AER and instantiate a `ev::vEvent` easily, as event decoding is provided by each

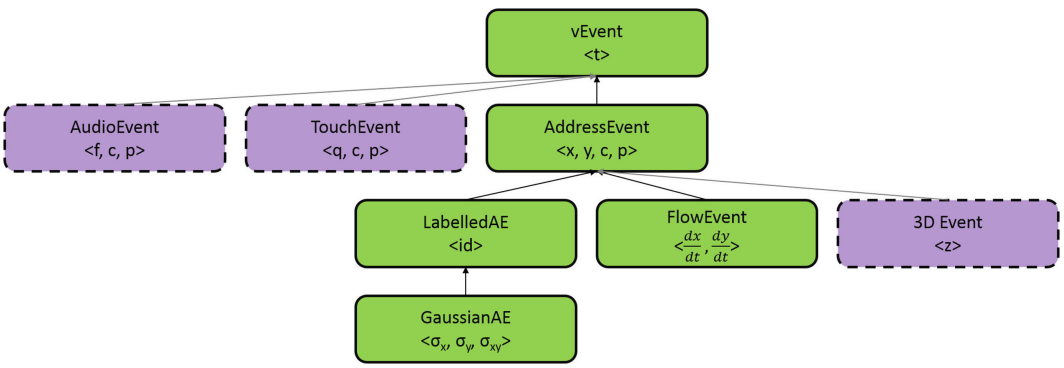

FIGURE 2 | Event-types and inheritance, purple/dashed boxes show possible additions to the library to support the integration of other sensory modalities and information from additional computing modules.

```
using namespace ev;

//create a new event in the centre of the ATIS sensor
event<AddressEvent> v1 = make_event<AddressEvent>();
v1->stamp = 0; v1->x = 152; v1->y = 120;

//upgrade the event to a labelledAE with a label of 1 (we make use of the
   keyword: auto)
auto v2 = make_event<labelledAE>(v1);
v2->id = 1;

//recast the event dynamically (using AE shorthand for AddressEvent)
auto v3 = as_event<AE>(v2);

//or recast statically (faster but less safe)
auto v4 = is_event<AE>(v2);
```

LISTING 1 | Instantiating events using shared pointer wrappers and dynamic casting. The outcome of the code-snippet will be the allocation of `v1` as an `ev::AddressEvent` and (an identical) `v2` as a `ev::labelledAE`, while `v3` and `v4` will be pointers to `v2`, but interpreted as `ev::AddressEvents`.

event class. Encoding/decoding typically involves bit-shifts and a typecast to interpret a specific range of bits as the correct data type. The decoded events are stored in a `ev::vQueue` which wraps a `std::deque<event<vEvent>>`. The procedure to obtain the event-stream is, therefore, transparent to the processing module. Reading `ev::vBottle` from a port is typically done using callback functionality (i.e., only where data is present) as the event-stream is asynchronous. An example code-snippet is provided **in Listing 2**.

Events can be saved and loaded from a file using the standard tools in YARP as an event-packet is fully interpretable as a standard `yarp::os::Bottle`. Therefore, it is easy to save a dataset using the `yarpdatadumper` and replay it using the `yarpdataplayer`. This is done externally to the event-driven library, simply by connecting the event-stream to/from the aforementioned modules using YARP connections.

3.3. Structuring the Event-Stream

The desired approach to ED processing is to perform a small, lightweight computation as each event is received; however, a single event (a single pixel) does not provide sufficient information on its own for many complex visual algorithms. Often it is necessary to store a sequence of events in order to extract useful information from their spatiotemporal structure. The type of structure used depends on the conditions, limitations and assumptions of the task or algorithm. For example, the length (in time) of a *ev::Temporal Window* can be tuned to respond to a target object moving at a certain velocity, but may fail if the target's velocity cannot be constrained. A *ev::Fixed Surface* of *N* events will be invariant to the speed of an object, but can fail if the target size and shape are unknown, a *ev::Surface* can access a spatial region-of-interest faster than a *ev::Temporal Window*, as long as the temporal order of events is not important. The event-driven library includes a range of methods to organize and structure the event-stream; an example code-snippet that combines port-callback functionality, event-packet decoding and event data structuring is shown in **Listing 2**.

In a distributed processing network, network latency, packet loss, and module synchronization become relevant issues, especially when it is desirable to take advantage of the intrinsic low-latency of the sensor. Processing needs to be performed in real-time to ensure robot behavior is decided from an up-to-date estimation of the world. The ATIS cameras will produce ≈10 kV/s when a small object is moving in the field of view but will produce >1,000 kV/s if the camera itself is rotated quickly (e.g., when the iCub performs a saccade). These numbers double for a stereo camera set-up. Real-time constraints can be broken if processing algorithms are dependent on processing every single event and the processing power is not sufficient.

```
using namespace ev;

class vManager : public yarp::os::BufferedPort<ev::vBottle> {

   ev::vSurface surface;

   //define the callback function of a yarp::os::BufferedPort<ev::vBottle>
      in the derived class "vManager"
   void vManager::onRead(ev::vBottle &inputBottle)
   {
      //decode only AddressEvents into a ev::vQueue
      vQueue q = inputBottle.get<AE>();

      //iterate over the events in the packet
      for(vQueue::iterator qi = q.begin(); qi != q.end(); qi++)
      {
         //static_ptr_cast to an ev::AddressEvent
         auto v = is_event<AE>(*qi);
         surface.addEvent(v);
      }

      //get the list of events on a 3x3 surface around the most recent
         event added to the surface
      ev::vQueue surfaceList = surface.getSurf(1);
   }
};
```

LISTING 2 | An example class for reading, decoding, and structuring events. This code will produce a small "surface" of events decoded from the AER representation automatically using the `ev::vBottle::get()` command, and the `ev::vBottle` is read asynchronously as the packets arrive.

In the YARP event-driven library, a multi-threaded event structure is provided to de-couple the process of reading events into a data structure from that of running the algorithm. Modules are constructed such that the entire history of events is accounted for, but the processing algorithm runs only at the rate at which it maintains real-time operation. The result is that chunks of events are not *randomly* lost within the communication pipeline; instead the rate at which the algorithm can output a result is reduced under high event-load. Our algorithms still typically run at rates of 100 to 1,000 s of Hertz; higher than the frame-rate of a standard camera. Importantly, the algorithm update-rate is not bottlenecked by the sensor update-rate (e.g., a frame-based camera), and the update-rate can be increased by adding computational power. The library classes `ev::queueAllocator`, `ev::tWinThread` and `ev::hSurfThread` manage real-time operation, and examples can be found in the documentation.

"Event-by-event" processing is also always possible in the YARP event-driven library and can be used to enforce a deterministic result to evaluate algorithm performance off-line, without the need to consider real-time constraints.

3.4. Low-Level Processing

Processing modules take the raw AER data and extract useful, higher-level information. The output of the modules will be a stream of events augmented with the additional information, as in **Figure 2**. The modules currently available in the event-driven repository are:

- Optical Flow—an estimate of visual flow velocity is given by the rate at which the position of events change over time. Local velocity can be extracted by fitting planes to the resulting spatiotemporal manifolds (Benosman et al., 2014). The vFlow module converts the ED camera output `ev::AddressEvent` to `ev::FlowEvent`.
- Cluster Tracking—The movement of an object across the visual field of an ED camera produces a detailed, unbroken trace of events. This module tracks clusters of events that belong to the same trace (Valeiras et al., 2015). The cluster center and distribution statistics is output from the `vCluster` module as a `ev::GaussianAE` event.
- Corner Detection—using an event-driven Harris algorithm, the event-stream is filtered to leave only the events on the corners of objects (Vasco et al., 2016a). Corner events provide unique features that can be tracked over time. Compared to a traditional camera, the ED corner algorithm requires less processing, as shown in **Figure 3B**. Corner events are represented by `ev::LabelledAE`s.
- Circle Detection—detection of circular shapes in the event-stream can be performed using an ED Hough transform. As the camera moves on a robot, many background events clutter the detection algorithm. The `vCircle` module reduces the false positive detections by incorporating optical-flow information (Glover and Bartolozzi, 2016). The detection results are shown in **Figure 3A**. Circle events are described by `ev::GaussianAE`s.
- Particle filtering—the particle filter achieves tracking that is robust to variations in the speed of the target, by also sampling within the temporal dimension (Glover and Bartolozzi, 2017). Ball tracking is implemented and the results are shown in **Figure 3A**.

The library also includes additional tools for:

- Camera Calibration—the intrinsic parameters of the camera can be estimated using a static fiducial and standard visual geometry techniques.
- Pre-processing—this module can apply a salt-and-pepper filter, flipping horizontal/vertical axes, applying camera distortion removal, and splitting a combined stereo event-stream into a left stream and a right stream.
- Visualization—the event-stream is asynchronous and does not inherently form "images" that can be viewed in the same way as a traditional camera. The `vFramer` uses a `ev::TemporalWindow` to accumulate events over time and produce a visualization of the event-stream. Different drawing methods exist for different event-types, which can be overlaid onto a single image (as shown in **Figure 1**).

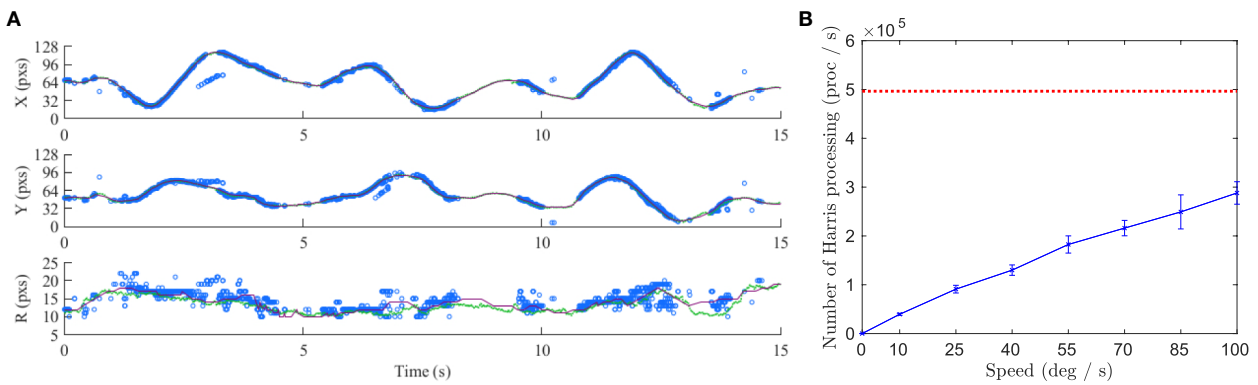

FIGURE 3 | The **(A)** tracking/detection accuracy of `ev::vCircle` (blue) and `ev::vParticleFiler` (green) compared to ground truth (black). Both algorithms can be used to control the iCub in gaze and grasping demonstrations. The **(B)** computation performance comparison between `ev::vCorner` and hypothetical frame-based Harris corner detection (red dashed line). A lower computational requirement is beneficial to resource contained on-line robotics. These figures can be generated using the scripts found in the datasets section.

4. DEMONSTRATIONS, CODE, AND DATASETS

The neuromorphic iCub and event-driven library have been used for several studies and robot demonstrations that can be run using `yarpmanager`. The modules are designed such that the robot begins performing the task once all required modules are running and the port connections have been made. Detailed instructions on how to run the demonstrations are provided in the online documentation[2] available with the code[3] on GitHub. An `xml` file is provided for each application to correctly launch and connect modules in `yarpmanager`. Known issues with the applications can also be found online. An overview of some of the applications is given below:

- **Ball Gazing and Grasping**—The module `vCircle` (desribed more in Glover and Bartolozzi (2016)) or `vParticleFilter` (described more in Glover and Bartolozzi (2017)) can be used to produce events describing the visual position of a ball, e.g., see **Figure 3A**. The `vGazeDemo` uses the `iKinGazeCtrl` (Roncone et al., 2016) to calculate the 3D position of the ball and the focus of the iCub's gaze can be directed to the location using the head and eye motors. Alternatively, the output of the ball position can be sent to the classic `DemoRedBall`[4] application to have the robot also move the arm to grasp the ball.
- **Stereo Vergence**—Automatic control of stereo vergence of the iCub to focus on an object within the field of view was implemented using biologically inspired methods (Vasco et al., 2016b). The `vVergence` application accepts stereo ev::AddressEvents and moves the vergence to minimize the response of stereo Gabor filters.
- **Attention and Micro-saccade**—A simple, yet effective, attention module is demonstrated that only requires the presence of events to perform a saccade to gaze at an external stimulus. If the event-rate is instead below a threshold, the `autosaccade` application generates small eye movements to visualize the static scene.

The documentation includes a project overview, instructions to run demonstration applications, descriptions and parameters of processing modules, and class and function descriptions. The code is only dependent on YARP and uses the `iCubContrib` to install the project in a manner compatible with YARP and iCub environment.

[2]http://robotology.github.io/event-driven/doxygen/doc/html/index.html.
[3]https://github.com/robotology/event-driven.
[4]https://github.com/robotology/icub-basic-demos.

Datasets of event-driven data can be found in the tutorials section of the online documentation. The datasets consist of the event-streams used in several of the experiments presented in this paper. The datasets enable the processing of event-driven algorithms if a physical camera is not available.

5. CONCLUSION

This paper presents the YARP-integrated event-driven library, specifically toward enabling ED robotics using a robot middleware. The data structures, multi-threaded approach and algorithm design are aimed toward real-time operation under a wide range of conditions and in uncontrolled environment, toward robust robotic behavior. The paper has presented the YARP interface, the low-level vision algorithms, and the applications on the iCub robot.

Event cameras are now available as an add-on plug-in and new iCub robots can potentially come equipped with neuromorphic hardware; alongside traditional cameras, or as the sole form of vision. Alternatively, the software package can be used through a USB interface to the ATIS camera, or through off-line datasets. The contribution of alternative camera interfaces is possible (and welcome) as the processing modules are transparent to the source of the events, and the package is provided as an open-source project.

AUTHOR CONTRIBUTIONS

All authors contributed to the writing and proofing of the article, as well as documentation of the code. CB was the major contributor to the hardware interfaces and AG was the major contributor to the libraries and applications. VV and MI contributed to modules and applications.

ACKNOWLEDGMENTS

The authors would like to thank Ugo Pattacini, Charles Clerq, and Francesco Rea for early contributions to the event-driven libraries, and Francesco Diotalevi, Marco Maggiali, and Andrea Mura for hardware and FPGA development, and for the integration of event cameras on the iCub.

FUNDING

This research has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 231467 (eMorph) and no. 284553 (SICODE).

REFERENCES

Bartolozzi, C., Rea, F., Clercq, C., Hofstätter, M., Fasnacht, D., Indiveri, G., et al. (2011). "Embedded neuromorphic vision for humanoid robots," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)* (Colorado Springs, CO: IEEE), 129–135.

Benosman, R., Clercq, C., Lagorce, X., Ieng, S.-H., and Bartolozzi, C. (2014). Event-based visual flow. *IEEE Trans. Neural Netw. Learn. Syst.* 25, 407–417. doi:10.1109/TNNLS.2013.2273537

Conradt, J., Cook, M., Berner, R., Lichtsteiner, P., Douglas, R. J., and Delbruck, T. (2009). "A pencil balancing robot using a pair of AER dynamic vision sensors," in *IEEE International Symposium on Circuits and Systems* (Taipei, Taiwan), 781–784.

Delbruck, T. (2008). "Frame-free dynamic digital vision," in *Proceedings of International Symposium on Secure-Life Electronics, Advanced Electronics for Quality Life and Society* (Tokyo, Japan), 21–26.

Delbruck, T., and Lang, M. (2013). Robotic goalie with 3 ms reaction time at 4% CPU load using event-based dynamic vision sensor. *Front. Neurosci.* 7:223. doi:10.3389/fnins.2013.00223

Glover, A., and Bartolozzi, C. (2016). "Event-driven ball detection and gaze fixation in clutter," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (Daejeon, South Korea), 2203–2208.

Glover, A., and Bartolozzi, C. (2017). "Robust visual tracking with a freely-moving event camera," in *IEEE International Conference on Intelligent Robots and Systems* (Vancouver, Canada: IEEE).

Hoffmann, R., Weikersdorfer, D., and Conradt, J. (2013). "Autonomous indoor exploration with an event-based visual SLAM system," in *European Conference on Mobile Robots, ECMR 2013 – Conference Proceedings* (Barcelona, Spain), 38–43.

Kim, H., Leutenegger, S., and Davison, A. J. (2016). "Real-time 3D reconstruction and 6-DoF tracking with and event camera," in *European Conference on Computer Vision*, Amsterdam, 349–364.

Kueng, B., Mueggler, E., Gallego, G., and Scaramuzza, D. (2016). "Low-latency visual odometry using event-based feature tracks," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (Daejeon, South Korea).

Lichtsteiner, P., Posch, C., and Delbruck, T. (2008). An 128x128 120dB 15μs-latency temporal contrast vision sensor. *IEEE J. Solid State Circuits* 43, 566–576. doi:10.1109/JSSC.2007.914337

Metta, G., Fitzpatrick, P., and Natale, L. (2006). YARP: yet another robot platform. *Int. J. Adv. Robot. Syst.* 3, 043–048. doi:10.5772/5761

Mortara, A. (1998). "A pulsed communication/computation framework for analog VLSI perceptive systems," in *Neuromorphic Systems Engineering*, ed. T. Lande (Norwell, MA: Kluwer Academic), 217–228.

Mueggler, E., Gallego, G., and Scaramuzza, D. (2015). "Continuous-time trajectory estimation for event-based vision sensors," in *Proceedings of Robotics: Science and Systems*, Rome. doi:10.15607/RSS.2015.XI.036

Ni, Z., Bolopion, A., Agnus, J., Benosman, R., and Régnier, S. (2012). Asynchronous event-based visual shape tracking for stable haptic feedback in microrobotics. *IEEE Trans. Robot.* 28, 1081–1089. doi:10.1109/TRO.2012.2198930

Posch, C., Matolin, D., and Wohlgenannt, R. (2011). A QVGA 143 dB dynamic range frame-free PWM image sensor with lossless pixel-level video compression and time-domain CDS. *IEEE J. Solid State Circuits* 46, 259–275. doi:10.1109/JSSC.2010.2085952

Rea, F., Metta, G., and Bartolozzi, C. (2013). Event-driven visual attention for the humanoid robot iCub. *Front. Neurosci.* 7:234. doi:10.3389/fnins.2013.00234

Rebecq, H., Horstschaefer, T., Gallego, G., and Scaramuzza, D. (2016). EVO: a geometric approach to event-based 6-DoF parallel tracking and mapping in real-time. *IEEE Robot. Autom. Lett.* 2, 593–600. doi:10.1109/LRA.2016.2645143

Roncone, A., Pattacini, U., Metta, G., and Natale, L. (2016). "A cartesian 6-DoF gaze controller for humanoid robots," in *Proceedings of Robotics: Science and Systems*, AnnArbor. doi:10.15607/RSS.2016.XII.022

Serrano-Gotarredona, T., and Linares-Barranco, B. (2015). Poker-DVS and MNIST-DVS. Their history, how they were made, and other details. *Front. Neurosci.* 9:481. doi:10.3389/fnins.2015.00481

Valeiras, D. R., Lagorce, X., Clady, X., Bartolozzi, C., Ieng, S.-H., and Benosman, R. (2015). "An asynchronous neuromorphic event-driven visual part-based shape tracking," in *IEEE Transactions on Neural Networks and Learning Systems*, 1–15. Available at: http://ieeexplore.ieee.org/document/7063246/

Vasco, V., Glover, A., and Bartolozzi, C. (2016a). "Fast event-based Harris corner detection exploiting the advantages of event-driven cameras," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (Daejeon, South Korea), 4144–4149.

Vasco, V., Glover, A., Tirupachuri, Y., Solari, F., Chessa, M., and Bartolozzi, C. (2016b). "Vergence control with a neuromorphic iCub," in *IEEE-RAS International Conference on Humanoid Robots (Humanoids)* (Cancun, Mexico: IEEE), 732–738.

Conflict of Interest Statement: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.