A Hybrid Neuromorphic Object Tracking and Classification Framework for Real-Time Systems

Ussa, Andrés; Rajen, Chockalingam Senthil; Pulluri, Tarun; Singla, Deepak; Acharya, Jyotibdha; Chuanrong, Gideon Fu; Basu, Arindam; Ramesh, Bharath

Published in:
IEEE Transactions on Neural Networks and Learning Systems

Published: 01/08/2024

Document Version:
Post-print, also known as Accepted Author Manuscript, Peer-reviewed or Author Final version

Publication record in CityU Scholars:
Go to record

Published version (DOI):
10.1109/TNNLS.2023.3243679

Publication details:
Ussa, A., Rajen, C. S., Pulluri, T., Singla, D., Acharya, J., Chuanrong, G. F., Basu, A., & Ramesh, B. (2024). A Hybrid Neuromorphic Object Tracking and Classification Framework for Real-Time Systems. IEEE Transactions on Neural Networks and Learning Systems, 35(8), 10726-10735. https://doi.org/10.1109/TNNLS.2023.3243679

Citing this paper
Please note that where the full-text provided on CityU Scholars is the Post-print version (also known as Accepted Author Manuscript, Peer-reviewed or Author Final version), it may differ from the Final Published version. When citing, ensure that you check and use the publisher's definitive version for pagination and other details.

General rights
Copyright for the publications made accessible via the CityU Scholars portal is retained by the author(s) and/or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights. Users may not further distribute the material or use it for any profit-making activity or commercial gain.

Publisher permission
Permission for previously published items are in accordance with publisher's copyright policies sourced from the SHERPA RoMEO database. Links to full text versions (either Published or Post-print) are only available if corresponding publishers allow open access.

Take down policy
Contact lbscholars@cityu.edu.hk if you believe that this document breaches copyright and provide us with details. We will remove access to the work immediately and investigate your claim.

Download date: 20/01/2025
Ussa, A., Rajen, C. S., Pulluri, T., & Singla, D. et al. (2023). A Hybrid Neuromorphic Object Tracking and Classification Framework for Real-Time Systems. *IEEE Transactions on Neural Networks and Learning Systems*. Advance online publication. https://doi.org/10.1109/TNNLS.2023.3243679
A Hybrid Neuromorphic Object Tracking and Classification Framework for Real-time Systems

Andrés Ussa, Chockalingam Senthil Rajen, Tarun Pulluri, Deepak Singla, Member, IEEE, Jyotibdha Acharya, Gideon Fu Chuanrong, Arindam Basu, Senior Member, IEEE, and Bharath Ramesh*, Member, IEEE

Abstract—Deep learning inference that needs to largely take place on the ‘edge’ is a highly computational and memory-intensive workload, making it intractable for low-power, embedded platforms such as mobile nodes and remote security applications. To address this challenge, this paper proposes a real-time, hybrid neuromorphic framework for object tracking and classification using event-based cameras that possess desirable properties such as low-power consumption (5 − 14mW) and high dynamic range (120dB). Nonetheless, unlike traditional approaches of using event-by-event processing, this work uses a mixed frame and event approach to get energy savings with high performance.

Using a frame-based region proposal method based on the density of foreground events, a hardware-friendly object tracking scheme is implemented using the apparent object velocity while tackling occlusion scenarios. The frame-based object track input is converted back to spikes for TrueNorth classification via the energy-efficient deep network (EEDN) pipeline. Using originally collected datasets, we train the TrueNorth model on the hardware track outputs, instead of using ground truth object locations as commonly done, and demonstrate the ability of our system to handle practical surveillance scenarios. As an alternative tracker paradigm, we also propose a continuous-time tracker with C++ implementation where each event is processed individually, which better exploits the low latency and asynchronous nature of neuromorphic vision sensors. Subsequently, we extensively compare the proposed methodologies to state-of-the-art event-based and frame-based methods for object tracking and classification, and demonstrate the use case of our neuromorphic approach for real-time and embedded applications without sacrificing performance. Finally, we also showcase the efficacy of the proposed neuromorphic system to a standard RGB camera setup when simultaneously evaluated over several hours of traffic recordings.

Index Terms—Event-based vision, object tracking, object classification, neuromorphic vision, FPGA implementation, IBM TrueNorth.

Source code: https://github.com/nusneuromorphic/eEOT

I. INTRODUCTION

As an emerging alternative to standard cameras, event cameras acquire information of a scene in an asynchronous and pixel independent manner, where each of them react and transmit data only when intensity variation is observed. This provides a steady stream of events with a very high temporal resolution (microsecond) at low-power (5 − 14mW), reducing redundancy in the data with improved dynamic range due to the local processing paradigm [1]−[3]. In particular, there is no significant need for background modeling, since a static event camera will only generate events corresponding to moving objects, thereby naturally facilitating tracker initialization. All these features are well suited for visual tracking applications but demand the use of algorithms designed to handle asynchronous events.

An event-by-event approach is dominantly seen in the literature for object tracking and detection using neuromorphic vision sensors [4]–[8]. The aim of these methods is to create an object representation based on a set of incoming events and updating it dynamically when events are triggered. Although these methods can be effective for specific applications, they often require high parametrization [4], [5], [8] or are not effective for tracking multiple objects [6].

Similar to the above works, [9] is an event-by-event approach for object tracking applications that performs a continuous event-based estimation of velocity using a Bayesian descriptor. Another example is [10], which proposes event-based tracking and detection for general scenes using a discriminative classification system and a sliding window approach. While these methods work intuitively for objects with different shapes and sizes, and even obtain good tracking results, they have not been implemented under a real-time operation requirement.

In contrast to the above methods, an aggregation of incoming events can be considered at fixed intervals instead of processing events as they arrive. This produces a more obvious representation of the scene (a “frame”), and allows an easier coupling with traditional feature extraction and classification approaches [12]–[14]. In [15], asynchronous event data is captured at different time intervals, such as 10ms and 20ms, to obtain relevant motion and salient information. Then, clustering algorithms and Kalman filter are applied for detection and tracking, achieving good performance under limited settings. Other examples of event-based frames along with variations in sampling frequency and recognition techniques are [14], [16], [17], which show the potential of this approach for detection.

Few recent works have made efforts to create embedded systems using these novel event-based sensors [18]–[22]. Taking this important step forward for real-time and embedded applications, we leverage the low-latency and high dynamic range of event cameras by directly interfacing to an FPGA processor. This allows to perform embedded object tracking...
Fig. 1: Block diagram of the hybrid real-time neuromorphic surveillance system. The event frame is fed to a median noise filtering module before passing it to the overlap-based tracker. A Poisson model of spike generation is then used to convert the track instances back to spikes. The neuromorphic processor [11] receives the track outputs and classifies it in an event-based manner using a deep spiking neural network.

Fig. 1 illustrates the proposed hybrid real-time neuromorphic framework. In essence, the focus of our approach is to build a real-time and embedded system that takes advantage of a stationary event camera, thereby picking up only moving objects and not being specific to background conditions. To this end, we use a hybrid approach that is different from purely event-based or purely frame-based approaches. First, the asynchronous events are accumulated into a binary image and an overlap-based tracking is performed on these frames. For subsequent object classification, the frames are converted back to spikes for efficient processing on the IBM neuromorphic chip.

As shown in the experiments later, our hardware-friendly trackers perform significantly better and requires far less resources (7×less memory and 3×less computations) than the popular multi-object event-based mean shift (EBMS) tracker [22] and the state-of-the-art frame-based SiamMask tracker [24]. The use of deep neural networks, specifically one-shot learning using Siamese architectures, has demonstrated success object tracking tasks [25], [26], as they can efficiently perform online tracking using similarity matching. Additionally, we compare the performance of the proposed neuromorphic system to a standard RGB camera setup when simultaneously evaluated over several hours of traffic recordings. This is of immense importance when using our fully embedded system for remote surveillance applications where long battery life of the sensor node is critical without sacrificing performance.

This paper is an extended version of the work initially published in BMVC Workshops 2019 [27]. Novel contributions over [27] include the event-based tracker extension (Sec. I-B) and comparison to the state-of-the-art event and frame-based trackers. We have also evaluated the performance of the proposed tracker on recordings from various commercially available neuromorphic vision sensors, such as CeleX and DAVIS640. Additionally, an extensive comparison is made between the TrueNorth classification output to state-of-the-art classification frameworks, such as SLAYER [28] and pre-trained models via transfer learning (Section III-A) [29].

II. METHODOLOGY

The DAVIS camera events [30] are utilized through the formation of frames for the task of tracking vehicles and humans on an urban landscape. Thus, the tracker performance hinges on the ability to capture frames at a rate much faster than the dynamics of the scene, thereby taking advantage of the low-latency of event cameras. The frames obtained are median filtered and region proposals are extracted from two 1-D histograms along X and Y directions for tracking. The tracker uses centroids and Euclidean distances to monitor up to eight objects simultaneously, while classification is performed on these locations using IBM’s TrueNorth neuromorphic chip [31] to assign one of the following classes: cars, motorbikes, buses, trucks and humans. The filtering and region proposal of the tracker follows the existing method in [32] and this work additionally considers occlusion, track velocity calculation and smooth interpolation between two instances of tracking. The full system is embedded on FPGA hardware and interfaced to IBM’s TrueNorth chip.

A. Object Tracking

This work proposes a simple, hardware-friendly tracker, termed as events overlap tracker (EOT), consisting of a series of steps, namely: track assignment, merging and post-processing. The core function resides in the track assignment task, similar to a Kalman filter update, while the merging and post-processing steps deal with the occlusion and track assignment issues. Each track output is defined by a set of properties: (1) The top-left location of the tracked object (x and y coordinates); (2) The width and height of the tracked object (w and h); (3) The velocities vx and vy of the object; (4) The tracker state (free, tracking, or locked) and (5) A unique ID. The free state indicates that the tracker is in stand-by and no region is currently assigned to it. The tracking state indicates that the tracker has matched with a region proposal once. The locked state indicates that the tracker has matched a region proposal in at least two consecutive frames and is currently “locked” on an object. Since only locked trackers are classified, having a tracking state allows to filter noisy tracks and reduce the burden on the classifier.

1) Track Assignment: The EOT track assignment procedure can be briefly summarized as follows. As a region proposal,
defined by its coordinates, \( r_{j}^{\text{new}} = \{x_{j}^{\text{new}}, y_{j}^{\text{new}}, w_{j}^{\text{new}}, h_{j}^{\text{new}}\} \), is received as input, its overlap area with respect to all active trackers \( T_{j}^{k} = \{x_{j}^{k}, y_{j}^{k}, w_{j}^{k}, h_{j}^{k}\} \) is measured, where \( k = 1, \cdots, N \) indicates the track IDs and \( j \) the frame instance. If their overlap is higher than the \textit{track assignment ratio} \( O_{th} \), the region proposal is then assigned to the respective tracker ID. Otherwise, the region proposal is assigned to a free tracker.

To begin with the track assignment, the tracker’s new position is estimated based on its previous velocity, and the resulting region is evaluated against the region proposal. The assignment is evaluated based on the calculation of the overlap area \( O_{A} \) between the existing track region and a new region, as defined in \( \text{[7]} \).

\[
O_{A} = \left( \max(0, \min(x_{j}^{k} + w_{j}^{k}, x_{j}^{\text{new}} + w_{j}^{\text{new}}) - \max(x_{j}^{k}, x_{j}^{\text{new}})) \right) \times \left( \max(0, \min(y_{j}^{k} + h_{j}^{k}, y_{j}^{\text{new}} + h_{j}^{\text{new}}) - \max(y_{j}^{k}, y_{j}^{\text{new}})) \right)
\]

(1)

2) Tracker Update: In our EOT implementation, a tracker assignment is made when the overlapping area is higher than a \( O_{th} \) of 20\%. Subsequently, the tracker properties and state are updated. If the current state is \textit{tracking}, then it is updated to \textit{locked}, and if it was already in \textit{locked} state, it will remain as it is. After a successful assignment, each tracker region is updated using a weighted average as stated in \( \text{[2]} \), for each of the spatial elements, where \( \alpha \) is the weighting degree coefficient.

\[
T_{j}^{k} = (1 - \alpha) \cdot r_{j}^{\text{new}} + \alpha \cdot (T_{j-1}^{k} + v_{j-1}^{k} \cdot \Delta t)
\]

(2)

where \( v_{j-1}^{k} \) refers to the velocity of the track (in pixels/s) at previous frame instance \( j - 1 \), and \( \Delta t = t_{j} - t_{j-1} \). The velocity, shown in \( \text{[5]} \), is also then averaged analogous to the position update. Similarly, it is also applied for the y direction.

\[
v_{j}^{k}(x) = (1 - \alpha) \frac{(x_{j}^{\text{new}} - x_{j-1}^{k}) + (w_{j}^{\text{new}} - w_{j-1}^{k})}{\Delta t} + \alpha \cdot v_{j-1}^{k}
\]

(3)

During the tracks assignment, in cases where different region proposals are assigned to the same tracker or vice versa, a merging between the pertinent regions is applied. If more than one region proposal, \( r_{j,\text{new}1}^{\text{new}} \) and \( r_{j,\text{new2}}^{\text{new}} \), is assigned to the same tracker \( T_{j-1}^{k} \), non-maximal suppression is performed among the common regions and the tracker region, to group the rectangles and assign it to \( T_{j}^{k} \). On the other hand, if there is a region proposal that is assigned to more than one tracker, an occlusion check is performed among all the valid trackers.

3) Occlusion Model: Considering that the objects to be tracked display a wide range of sizes and often follow opposite directions or move at different speeds, the case in which an object occludes another occurs regularly. In occlusion scenarios, the event frame would show a bigger region than the individual objects without a clear boundary between them. In other words, the trackers under evaluation will overlap after one or two steps in the future based on the estimated velocity. Before an occlusion occurs, let us denote the implicated trackers as \( T_{j}^{a} \) and \( T_{j}^{b} \), and their size before occlusion as \( (w_{a}^{o}, h_{a}^{o}) \) and \( (w_{b}^{o}, h_{b}^{o}) \), respectively. Based on the trackers’ original sizes, their velocity, direction and the combined area after occlusion, it is possible to approximate their positions during the occluded frames. For this, a set of conditions is determined: trackers’ common direction \( cd = v_{j}^{a} + v_{j}^{b} > v_{j}^{a} \lor v_{j}^{b} \), width increase \( wi = (w_{j}^{a} > w_{j-1}^{a}) \) and highest velocity object \( hvo = abs(v_{j}^{a}) > abs(v_{j}^{b}) \).

While the occlusion is occurring, the change in width of the merged tracks, i.e. \( wi \), is used as the criteria to determine whether the affected tracks are coming together (\( wi = \text{False} \)) or getting apart (\( wi = \text{True} \)). In particular, for a tracker \( T_{j}^{a} \), it remains as the current region proposal when \( wi \) is \text{False}, \((x_{j}^{a}, y_{j}^{a}, w_{j}^{a}, h_{j}^{a}) \leftarrow (x_{j}^{\text{new}}, y_{j}^{\text{new}}, w_{j}^{\text{new}}, h_{j}^{\text{new}})\), or otherwise when \( wi \) is \text{True}, the track is equal to its original size in the region proposal, \((x_{j}^{a}, y_{j}^{a}, w_{j}^{a}, h_{j}^{a}) \leftarrow (x_{j}^{\text{new}} - w_{a}^{o} - w_{j}^{a}, y_{j}^{a} + h_{j}^{a} - h_{a}^{o}, w_{a}^{o}, h_{a}^{o})\). This situation occurs when the objects are moving in opposite directions (\( cd = \text{False} \)), or when they move in a common direction and the velocity of the tracker under evaluation, \( T_{j}^{a} \), is faster than its pair (\( hvo = \text{True} \)). On the other hand, if \( hvo = \text{False} \), then the track is intuitively set as the other component of the region proposal, \((x_{j}^{a}, y_{j}^{a}, w_{j}^{a}, h_{j}^{a}) \leftarrow (x_{j}^{\text{new}} - w_{a}^{o}, y_{j}^{a} + h_{j}^{a} - h_{a}^{o}, w_{a}^{o}, h_{a}^{o})\).

4) Cleanup: Finally, the post-processing step removes trackers that no longer match any of the region proposals. This is carried out by comparing the current state of the tracker with its past state. If a tracker was previously set as \text{locked} or \text{tracking} state, and in the current frame it no longer exists, then it is likely that the object is lost. However, since a region proposal can be inconsistent through time, due to the hardware noise from the DAVIS, it is important not to set the tracker free instantly. An intermediate maximum unlocks state is used to determine when a tracker is lost for several consecutive frames, and only in that case, it is set \text{free}. Additionally, an out-of-bounds check is performed to release trackers when objects leave the scene.

Note that although the EOT tracking is discontinuous in time, the location and size of the tracked object can be estimated continuously, allowing the size and location of the object to be determined in-between the frames. In other words, any time \( t \) satisfying \( t_{k}^{j} < t < t_{k+1}^{j} \),

\[
j = \arg \min_{i \geq t_{k}^{j}} (\lambda_{i})
\]

(4)

\[
\lambda = (t - t_{j-1}^{k})/(t_{k}^{j} - t_{j-1}^{k})
\]

\[
T = T_{j-1}^{k} + \lambda(T_{k}^{j} - T_{j-1}^{k})
\]

where \( j \) is the index of the \textit{closest track} and \( \lambda \) is the interpolation factor for time \( t \). Using the above equations, the location \((x, y)\) and size \((w, h)\) can be calculated for an interpolated track \( T = \{x, y, w, h\} \) at any time \( t \). This feature is useful for continuous-time EOT implementations as described next for certain applications.

B. Continuous-time EOT

An events overlap tracker in continuous-time is also presented using the fundamental concepts behind EOT while aiming to fully leverage the low latency nature of event-based cameras. The tracking stage processes each event individually and can be broken into two substages. The first substage
assigns each event to one of the trackers (or to no tracker), and
the second substage updates the assigned tracker using the new
event information and determines whether the tracker status
is active or inactive. Periodically, (typically 25ms) a cleanup
operation is performed to update old trackers and to merge
overlapping trackers. Finally, an occlusion check is performed
to improve tracking performance for objects overlapping each
other.

We define the ith tracker $T_i$ as

$$T_i = \{x_i, y_i, dx_i, dy_i, active_i, isi_i, t_i\}$$  (5)

where $x_i, y_i$ is the current tracker location, $dx_i, dy_i$ is the half-width and half-height of the rectangular tracker, $active_i$ is
boolean, indicating whether the tracker is active or not, $isi_i$ is
the average inter spike interval, and $t_i$ is the time at which
the tracker was last updated.

1) Track Assignment: For each event, the x-direction and
y-direction distances to each tracker center are computed and
compared to the tracker rectangle size. If the event lies within
the tracker rectangle for an active tracker, then it is assigned to
that active tracker. More formally, the event $e_j$ will be assigned to
the first tracker $T_j$ which is found such that

$$active_j = true$$

$$|x_j - x_i| \leq dx_j$$

$$|y_j - y_i| \leq dy_j$$  (6)

If no such tracker $T_j$ exists, then the event does not lie close
easy to any active tracker and will instead be assigned to
the nearest inactive tracker. More formally, $T_k$ according to

$$\min_{k|active_k=false} \sqrt{(x_k - x_i)^2 + (y_k - y_i)^2}$$  (7)

It is possible that an event gets matched to no trackers.
This can occur when all trackers are active, but the event is
not close enough to any of the active trackers to be allocated
to that one of them, in which case the event is omitted.

2) Tracker Update: The tracker update step consists of
updating the assigned tracker’s location, and keeping record of
the average time between events assigned to the tracker. These
quantities are updated using an exponential moving average.
If the tracker $T_j$ is being updated by event $e_i$, the position update takes the form

$$x_j \leftarrow \alpha x_j + (\alpha - 1)x_i$$

$$y_j \leftarrow \alpha y_j + (\alpha - 1)y_i$$  (8)

where $\alpha$ is typically set to 0.95.

If the event lies within the tracker region, as determined by
satisfying the last two conditions in (6), then the inter spike
interval is updated as

$$isi_j \leftarrow \alpha isi_j + (1 - \alpha)(t_i - t_j)$$  (9)

where $\alpha_i$ is typically set to 0.9.

The time $t_j$ is also modified according to the tracker’s last
update as $t_j \leftarrow t_i$.

Finally, we perform a check to ascertain whether the tracker
should be considered active or not. The check compares
the average inter spike interval per pixel within the tracker
rectangle to a fixed threshold. If the interval is small enough,
the tracker is considered active. The average is calculated as

$$isi_j \times dx_j \times dy_j$$

If the quantity above is less than the active threshold
$\Theta_{active}$, then the tracker is marked as active by setting
$active_j \leftarrow true$, otherwise we mark the tracker as inactive
using $active_j \leftarrow false$.

3) Periodic Cleanup: A periodic cleanup of trackers is
performed every 25ms. Without the cleanup, trackers would
only be updated when events are assigned to them, which
leads to the possibility of an active tracker remaining active
definitely. The cleanup updates each tracker by generating
a false event on the same location as the tracker and at the
time of the cleanup, and following the same update process
as described in Section II-B2.

Once all the trackers are updated, a check is performed to
merge overlapping active trackers. Two trackers $T_i$ and $T_j$ are
merged only if

$$active_i = active_j = 1$$

$$|x_i - x_j| \leq dx_i + dx_j$$  (10)

$$|y_i - y_j| \leq dy_i + dy_j$$

The new tracker location is the mean of the location of the
two trackers being merged. The size of the merged tracker is
either the size of the larger tracker, or the sum of the sizes
of the two trackers, depending on whether the center of
the smaller tracker lies within the rectangular region defined by
the larger tracker. Once a tracker is merged into another, it is
randomly re-initialized.

4) Occlusion Model: An occlusion handling stage is also
implemented in this pipeline taking as reference the occlusion
check procedure used in Section II-A3. The first step is to
detect an actual occlusion is happening. As there is no concept
of a “region” in this purely event-based method, possible
occlusions are initially detected when an incoming event is
being matched to the available trackers (Section II-B2). If an
event gets matched to more than one active tracker, then the
trackers involved are considered for further checks.

The trackers under occlusion are compared against each
other to check for minimum conditions of velocity and co-
variance. This step allows to filter out false occlusions from
being processed. The velocity conditions are shown in (11) and
(12), and their purpose is to ensure there is a minimum velocity
difference between the trackers. The covariance condition is
shown in (13) and its purpose is to verify that the estimated
error of the trackers is low.

$$D_a = \{\|V_{x_i} - V_{x_j}\| > V_a \mid sign(V_{x_i}) = sign(V_{x_j})\}$$  (11)

$$D_\beta = \{\|V_{x_i} - V_{x_j}\| > V_\beta \mid sign(V_{x_i}) \neq sign(V_{x_j})\}$$  (12)

$$P_d = diag(P_i) < P_T \land diag(P_j) < P_T$$  (13)

where $V_a$ and $V_\beta$ are velocity threshold values for the same
direction and opposite direction cases, respectively, $P_T$ is the
covariance threshold value, and $diag(P_i)$ and $diag(P_j)$ are
the diagonal sum of the covariance of each tracker being
compared. Then, if $P_d \land D_a \lor D_\beta = 1$, the trackers concerned are
considered for the next occlusion detection step.
Following the occlusion detection logic from Section II-A3, the last action is to check for overlapping between the trackers at one or two steps in the future. However, there is no concept of “timestep” in this method since there is no frame creation. Therefore, for this purpose, an occlusion timestep $O_t$ was set to perform this verification. The position of a tracker box on subsequent timesteps is calculated as in (14).

$$\begin{align*}
x_i &= x_i + Vx_i \cdot O_t \cdot n \\
y_i &= y_i + Vy_i \cdot O_t \cdot n
\end{align*}$$

where $n$ is the number of timesteps to take into consideration. Then, if an overlap is found at either $n = 1$ or $n = 2$, the trackers are flagged as occluding.

While in occlusion state, the processing of occluding trackers changes. First, while in normal state one incoming event only updates a single tracker, in occlusion state both implicated trackers are updated based on a single matching event. Second, the box size of the involved trackers is preserved throughout the occlusion. This keeps the boxes from over expanding due to the increase of events in the proximity result of the overlapping. Lastly, the velocity of the trackers before occlusion is kept fixed during the occlusion period. This is used to estimate the motion of the tracks during occlusion and to attempt to recover the locked objects after separation.

It is worth mentioning that the occlusion detection process explained above is applied even on trackers marked as occluding. Then, when evaluating (14), if the occluding trackers no longer show to be overlapping at future timesteps, their flag is then removed and the occlusion is assumed concluded.

5) Implementation: A C++ implementation of the tracker was written and evaluated. It is capable of performing tracking far faster than real-time, allowing us to generate tracking information for the entire dataset in approximately 15 mins. The C++ tracker output was written to a file and was used to generate the results shown later.

Subsequent object classification on Truenorth and other details of the Trenz-Zynq hardware implementation are as in the earlier work [27]. It is worth pointing out the power consumption of our system is about 550mW, which is $3 \times$ lower than performing inference on the edge for a similar deep learning network. In particular, an Inception-v3 network running on Google’s edge TPU in operation consumes about 1.7W [33], [34].

III. EXPERIMENTS

The development of this work demanded the acquisition of event-based data from a real application scenario for the purposes of training, validation and testing of our system. The main requirement for these recordings was a high, perpendicular view from the road near intersections. Under this condition, three places inside our campus were chosen for data recording. Fig. 2a shows a side-on recording at 3pm with a 12mm lens. Fig. 2b shows another 3pm recording at a different location with a 6mm lens. Lastly, Fig. 2c shows a cluttered setting recorded at 6pm with a 45mm lens. The events are aggregated to generate a frame either every 66 ms. During trial-and-error experiments, surprisingly even longer time periods (100ms) did not degrade system performance, although it increases the latency of the system significantly. The setting of 66 ms for generating the frame was chosen as a trade-off between accuracy and latency.

Table I shows the distribution of the collected dataset in terms of the number of samples obtained for each category and the distribution of the training/test set. We noticed that there were a lot of car samples, and thus to balance the training data, the samples were augmented by random flipping, rescaling (up to 140%) and rotation (up to 20 degrees in either direction). After augmenting a sample, it was cropped back to $42 \times 42$ pixels for training with a fixed image size on TrueNorth. A separate test dataset captured at a different time was used for evaluating the system. Among a total of 13 recordings, ten were used for training and three were used for testing, roughly 75% recordings for training and 25% for testing.

Fig. 2: Examples of recorded event-based traffic data shown alongside a RGB sensor recording. The figures correspond to the three sites (a) Side-on 70m away from the road; (b) top-down 30m away and (c) Side-on angle 20m away in a cluttered setting.
TABLE I: Number of samples per category in the collected dataset and the percentage of samples per category used for training.

| Site   | Car  | Bus | Pedestrian | Bike | Truck/Van |
|--------|------|-----|------------|------|-----------|
| 1      | 322  | 30  | 115        | 43   | 18        |
| 2      | 226  | 105 | 53         | 14   | 28        |
| 3      | 390  | 181 | 89         | 39   | 56        |
| Sum    | 938  | 316 | 257        | 96   | 102       |
| Overall| 54.89| 18.49| 15.04      | 5.62 | 5.97      |
| Train% | 67.05| 80.89| 74.41      | 74.24| 63.43     |
| Test%  | 32.95| 19.11| 25.59      | 25.76| 36.57     |

A. Comparison to State-of-the-art

In this section, we first report the performance of the proposed EOT, Continuous-EOT trackers and compare it to the popular multi-object event-based mean shift (EBMS) tracker [23], SiamMask [24] and conventional Kalman filter (KF) tracker. Additionally, we also evaluate our tracker performance on recordings from different neuromorphic vision sensors. Next, we compare the classification performance of the TrueNorth model against state-of-the-art method DART [6] and SLAYER [28] to investigate whether there is a performance drop due to the binary frame generation process in our neuromorphic framework. For a direct comparison between events and RGB data, we report the tracking and classification performance on simultaneously recorded RGB data compared to events. Finally, we show how the TN EEDN hyperparameter constraints affect the classification performance compared to fully-trained CNNs on the ImageNet database via transfer learning.

1) Comparison of Tracker Performance: To analyze the effect of finite bit precision in hardware implementations, we extracted the region proposals from the FPGA and passed it through the software tracker to generate precision and recall curves at the different intersection over union (IoU) thresholds, following the protocols in [32]. Fig. 3 compares the proposed EOT and the continuous-time EOT tracker performance to the multi-object event-based mean shift (EBMS) tracker [23], a frame-based deep learning tracker [24] and a Kalman Filter (KF) tracker used in [35]. For initializing the Kalman Filter tracker and SiamMask, an initial bounding-box location using the manual track annotations is given as input for each object. Note that in the Continuous-EOT tracker, there is no frame creation as the input to the tracker and is computationally faster.

It can be seen that the EOT tracker outperforms the multi-object EBMS tracker [23] comfortably (7× less memory and 3× less computations as noted in [32]). In terms of overall F1-score combining the precision and recall statistics, the EOT-software tracker scores 0.35 compared to 0.21 for the EBMS. Additionally, the EOT tracker performs better than the KF tracker applied to event-based binary frames. Note that EOT hardware has a slightly lower F1-score of 0.3 due to finite precision arithmetic. It can be deduced that the Continuous-EOT tracker, which processes each event individually, has similar performance to EOT in low IOUs and performs gradually worse as IOU increases. Since the latency is much lower, it can be potentially used as a real-time intrusion detection system for fast moving objects, as the frame-generation process in EOT is avoided. Following a similar analysis performed in [32], we note that Continuous-EOT requires 2× more memory compared to EOT due to explicit maintenance of inactive trackers. On the other hand, Continuous-EOT entails 3× less computations compared to EOT as there is no explicit region proposal step. The choice between these two are thus application-dependent.

Fig. 3 also compares the proposed EOT trackers to the deep learning based SiamMask [24] tracker, which is popular for its simplicity, speed and online learning. The precision of SiamMask is better than KF while being almost the same as the EBMS tracker. In terms of recall, SiamMask is only slightly better than KF while being significantly lower than EBMS. We observed that without an explicit occlusion model, SiamMask fails to track the object when another object crosses its path and this leads to a corruption of its online learning representation.
Results for detection probability are shown in Fig. 4b. All sensors present positive results for this metric, showing a probability above 60% for IoU values under 0.5. Further, the Celex sensor performs marginally better for most of the IoU values, confirming the intuition that a higher resolution sensor will have a higher probability of detecting an object in its field of view.

3) Comparison to state-of-the-art classification methods:
Table III shows the TrueNorth test accuracy evaluated using the ground truth (not using the tracker output) under two settings: per-sample (each instance of the object classified separately) and per-track (majority voting classification of the object instance from the time of entering the field-of-view and exiting). As expected, there is a significant improvement in the accuracy when considered on a per-track basis. Nonetheless, the TrueNorth model struggles with Trucks due to their similarity to both buses and cars, but does very well on all other classes, especially when given multiple opportunities to classify them on a per-track basis.

However, it is imperative that for a real-world surveillance application, the back-end classifier is trained on representative samples from the tracker output rather than on manually annotated ground truth tracks. The tracker output does not have object labels, and in order to train the TrueNorth classifier, class labels are automatically generated using their overlap with the ground truth object locations. The spurious tracks with no ground truth overlaps can be labelled as an additional background class and subsequently TrueNorth can classify them as false positives during deployment. Table III shows the TrueNorth classification accuracies obtained on the test track output (not the ground truth test tracks) by three models trained on: (1) ground truth track outputs, (2) augmented ground truth track outputs, and (3) the hardware tracker outputs using the training data. We see that the models trained on ground truth tracks that contain no background class perform poorly. This is an important highlight of our proposed system - to be able to respond to moving background conditions or spurious tracker outputs. In particular, the model trained on the tracker output gets a much higher accuracy (70.4%) compared to models trained on ground truth tracks.

In practice, the above-mentioned system accuracy will be higher due to limitations in evaluating the tracker outputs obtained by auto-generated class labels from the manually annotated ground truth. Trackers that do not have overlap with their target can only be labelled as background and this is especially true for the case of an object leaving/entering the scene when manual annotations do not exist. Thus, when the tracker locks on before the ground truth tracks have started, TrueNorth correctly classifies it as a bus, but since it does not agree with the annotation, it is marked as an error in the evaluation (Table III). Similarly, when the object exits the scene, TN correctly classifies it as a bus but it is counted as a statistical type II error because of the mismatch with the ground truth annotation. A demo of our fully embedded system reveals this scenario clearly.

1 Video demo (updated): https://tinyurl.com/ycc2tn5t
Table I compares the TrueNorth classification accuracy to the Distribution Aware Retinal Transform (DART) framework [6], which has obtained state-of-the-art accuracy on multiple event-based object datasets. It is worth stating that the DART method utilizes all the event information and obtains 95.4% per-track accuracy on the ground truth test dataset. This is understandably higher than the per-track 90.2% accuracy obtained using the EEDN framework, but more importantly, our approach shows that event cameras can be utilized to generate “frames” without sacrificing much performance for embedded surveillance applications.

Table II also compares the classification performance of TrueNorth and SOTA learning algorithm Spike layer error re-assignment in time (SLAYER) [28], which learns both weight and axonal delays in SNNs. The classification performance of SLAYER shows a similar trend as the DART method, higher than TrueNorth classification performance. It is interesting to observe that the difference in balanced accuracy mainly depends on the classifier performance on Trucks, whereas the difference in accuracy’s in other classes across various methods is negligible. In other words, fine-grained classification needs to exploit subtle inter-class object appearance variations, which can be captured by event-based methods exploiting high temporal resolution of the event camera.

4) Comparison with RGB: For a close comparison of the proposed EOT and Continuous-EOT trackers to the KF tracker [35], the method was tested on standard RGB data recorded as they were not developed for event-based binary frames. Fig. 5 compares the tracking performances on RGB data. Both Continuous-EOT and EOT tracker again outperforms the frame-based tracker with a higher F1-score of 0.35. However, the EOT tracker’s overall performance is slightly better, with an F1-score of 0.231 compared to 0.225 for Continuous-EOT. As expected, the Kalman Filter implementation performs better on RGB data when compared to event-based data. The KF-based multi-object tracker scores 0.119 on RGB and only 0.061 for events data.

5) Comparison to Pre-trained Models: RGB and Events ground truth data are streamed into pre-trained networks such as Alexnet [36], Resnet-18 and Resnet-50 [29] via transfer learning to see how TrueNorth performs compared to CNNs that have more number of layers and having been trained on over a million images. Recall that TrueNorth uses a 15-layer CNN with hardware constraints on the hyper-parameters [11]. Figure 7 shows a gradual increase in accuracy from Alexnet to Resnet-18 and Resnet-50 for both RGB as well as Events datasets. RGB performs slightly better than events, which can be attributed to having more visual information such as color, texture, etc. Nonetheless, TN achieves a comparable performance to pre-trained CNNs with deeper architectures on both events and RGB datasets.

| Method            | Car% | Bus% | Human% | Bike% | Truck% | Balanced% |
|-------------------|------|------|--------|-------|--------|-----------|
| TN(Per-Sample)    | 90.4 | 92.5 | 94.7   | 86.9  | 54.2   | 83.3      |
| TN (Per-Track)    | 99.0 | 98.2 | 100    | 100   | 53.8   | 90.2      |
| TN RGB            | 99.4 | 98.0 | 100    | 95.5  | 72.5   | 93.1      |
| DART (Per-Track)  | 96.3 | 96.6 | 100    | 100   | 83.9   | 95.4      |
| SLAYER (Per-Track)| 97.6 | 98.3 | 100    | 100   | 78.1   | 94.8      |
| RESNET-18 (Per-Track) [29] | 94.8 | 93.1 | 99.0 | 100 | 87.0 | 94.8 |
| RESNET-18 RGB     | 99.4 | 98.0 | 100    | 100   | 82.5   | 96.0      |

Table III: Type of Training Data vs. Per-Track TrueNorth Test Accuracy.

| Class Name | GT % | Aug. GT % | FPGA % |
|------------|------|-----------|--------|
| 1 Human    | 91.8 | 96.3      | 85.8   |
| 2 Bike     | 62.2 | 72.9      | 50.0   |
| 3 Car      | 62.0 | 56.7      | 89.8   |
| 4 Truck    | 22.7 | 35.1      | 28.5   |
| 5 Bus      | 53.1 | 54.0      | 84.2   |
| 6 Other    | 39.7 | 39.1      | 82.6   |
| Overall    | 48.7 | 52.5      | 70.4   |

To prove the classification performance on the dataset recorded by the event camera is in the same ballpark with dataset simultaneously recorded by standard RGB Camera, we compare the TrueNorth classification model performance on the events and RGB datasets. Note that the RGB dataset has the same train and test split as the events data. Fig. 6 shows the TN performance of Events and RGB 70×70 are in the same neighbourhood whereas RGB 42×42 shows a significant performance drop. This also implies that higher image resolution better the classification performance on TrueNorth. Augmented datasets show a slight increase in performance compared to unbalanced ones. In particular, the CNN inference on TrueNorth for a 70×70 RGB image utilizes 3721 out of the available 4096 cores, which is 1.4 times the number of cores used by the events data.
We have demonstrated a strong use case of our neuromorphic framework for real-time and embedded applications without sacrificing performance.

\section*{IV. Conclusion}

This paper presented one of the first end-to-end neuromorphic frameworks for real-time object tracking and classification, demonstrated using a low-power hardware implementation that consumes about 0.5W, which is 3x less power than conventional TPUs used for deep learning. The proposed framework employs a hybrid approach consisting of events aggregated into frames for maintaining individual track of objects in occluded scenarios. Subsequently, the tracked object was efficiently classified using the IBM EEDN pipeline of the spike-based neuromorphic chip. In this setup, the TrueNorth chip was time-multiplexed to handle eight objects while making sure that the neurons used for preprocessing will scale linearly with the number of objects to be pseudo-simultaneously classified. Using a continuous-time tracker implementation, we demonstrated the use case of the proposed event-based tracker to fully exploit the low latency characteristics of the event camera. In addition to a real-time demo, we extensively compared the proposed tracking and classification methods to state-of-the-art event-based and frame-based deep learning methods and showed its relevance to on-going work in the research field. We also demonstrated that the proposed neuromorphic system achieves better tracking and classification performance compared to a standard RGB camera setup when simultaneously evaluated over several hours of traffic recordings. In summary, we have demonstrated a strong use case of our neuromorphic framework for real-time and embedded applications without sacrificing performance.

\begin{thebibliography}{10}

[1] P. Lichtsteiner, C. Posch, and T. Delbruck, “A $128 \times 128$ 120 dB 15 $\mu$s latency asynchronous temporal contrast vision sensor,” IEEE Journal of Solid-State Circuits, vol. 43, no. 2, pp. 566–576, 2008.

[2] C. Posch, D. Matolin, R. Wohlgemann, M. Hofsttter, P. Schn, M. Litzenberger, D. Bauer, and H. Garn, “Live demonstration: Asynchronous time-based image sensor (atis) camera with full-custom aae processor,” in Proceedings of 2010 IEEE International Symposium on Circuits and Systems, 2010, pp. 1392–1392.

[3] S. Chen and M. Guo, “Live demonstration: Clex-v: A 1m pixel multimode event-based sensor,” in 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), 2019, pp. 1682–1683.

[4] X. Lagorce, C. Meyer, S.-H. Ieng, D. Filliat, and R. Benosman, “Asynchronous Event-Based Multikernel Algorithm for High-Speed Visual Features Tracking,” IEEE Transactions on Neural Networks and Learning Systems, pp. 1–12, Aug 2014.

[5] A. Glover and C. Bartolozzi, “Robust visual tracking with a freely-moving event camera,” in IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, Sep 2017, pp. 3769–3776.

[6] B. Ramesh, H. Yang, G. M. Orchard, N. A. Le Thi, S. Zhang, and C. Xiang, “DART: Distribution Aware Retinal Transform for Event-based Cameras,” IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 1–1, 2019.

[7] Q. Xu, Y. Li, J. Shen, P. Zhang, J. K. Liu, H. Tang, and G. Pan, “Hierarchical spiking-based model for efficient image classification with enhanced feature extraction and encoding,” IEEE Transactions on Neural Networks and Learning Systems, pp. 1–9, 2022.

[8] H. Chen, S. Lin, D. Suter, Y. Yan, and H. Wang, “Event data association via robust model fitting for event-based object tracking,” arXiv preprint arXiv:2110.12962, 2021.

[9] L. Dardelet, S.-H. Ieng, and R. Benosman, “Event-Based Features Selection and Tracking from Intertwined Estimation of Velocity and Generative Contours,” CoRR, Nov 2018.

[10] B. Ramesh, S. Zhang, Z. W. Lee, Z. Gao, G. Orchard, and C. Xiang, “Long-term object tracking with a moving event camera,” in BMVC, 2018.

[11] S. K. Esser, R. Appuswamy, P. Merolla, J. V. Arthur, and D. S. Modha, “Backpropagation for energy-efficient neuromorphic computing,” in Advances in Neural Information Processing Systems (NIPS). Curran Associates, Inc., 2015, vol. 28, pp. 1117–1125.

[12] Z. Ni, S.-H. Ieng, C. Posch, S. Régnier, and R. Benosman, “Visual Tracking Using Neuromorphic Asynchronous Event-Based Cameras,” Neural Computation, vol. 27, pp. 925–953, 2015.

[13] S. Banerjee, H. H. Chopp, J. Zhang, Z. W. Wang, P. Kang, O. Cossairt, and A. Katsaggelos, “A joint intensity-neuromorphic event imaging system with bandwidth-limited communication channel,” IEEE Transactions on Neural Networks and Learning Systems, pp. 1–15, 2022.

[14] M. Iacono, S. Weber, A. Glover, and C. Bartolozzi, “Towards Event-Driven Object Detection with Off-the-Shelf Deep Learning,” in 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). Madrid: IEEE, Oct 2018, pp. 1–9.

[15] G. Hinz, G. Chen, M. Aafaque, F. Röhrbein, J. Conradt, Z. Bing, Z. Qu, W. Stiechele, and A. Knoll, “Online Multi-object Tracking-by-Clustering for Intelligent Transportation System with Neuromorphic Vision Sensor,” in Ki 2017: Advances in Artificial Intelligence. Dortmund, Germany: Springer, Cham, 2017, pp. 142–154.

[16] Y. Alkendi, R. Azzam, A. Ayyad, S. Javed, L. Seneviratne, and Y. Zweiri, “Neuromorphic camera denoising using graph neural network-driven transformers,” IEEE Transactions on Neural Networks and Learning Systems, pp. 1–15, 2022.

[17] Y. Chae, L. Wang, and K.-J. Yoon, “Siamevent: Event-based object tracking via edge-aware similarity learning with siamese networks,” arXiv preprint arXiv:2109.13456, 2021.

\end{thebibliography}
[18] L. A. Camuas-Mesa, J. Luna-Barranco, and T. Serrano-Gotarredona, “Low-power hardware implementation of snn with decision block for recognition tasks,” in 2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS), 2019, pp. 73–76.

[19] C. Iaboni, H. Patel, D. Lobo, J.-W. Choi, and P. Abichandani, “Event camera based real-time detection and tracking of indoor ground robots,” IEEE Access, vol. 9, pp. 166588–166602, 2021.

[20] M. Macanovic, F. Chersi, F. Ruttard, S. Ieng, and R. Benosman, “When conventional machine learning meets neuromorphic engineering: Deep temporal networks (dtnets) a machine learning framework allowing to operate on events and frames and implantable on sensor flow like hardware,” CoRR, vol. abs/1811.05767, 2018.

[21] S.-C. Liu, B. Rueckauer, E. Ceolini, A. Huber, and T. Delbruck, “Event-driven sensing for efficient perception: Vision and audition algorithms,” IEEE Signal Processing Magazine, vol. 36, no. 6, pp. 29–37, 2019.

[22] Z. W. Wang, P. Duan, O. Cossairt, A. Katsaggelos, T. Huang, and B. Shi, “Joint filtering of intensity images and neuromorphic events for high-resolution noise-robust imaging,” in 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020, pp. 1606–1616.

[23] T. Delbruck and M. Lang, “Robotic goalie with 3 ms reaction time at 4% cpu load using event-based dynamic vision sensor,” Frontiers in Neuroscience, vol. 7, p. 223, 2013.

[24] Q. Wang, L. Zhang, L. Bertinetto, W. Hu, and P. H. Torr, “Fast online object tracking and segmentation: A unifying approach,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 1328–1338.

[25] G. R. Koch, “Siamese Neural Networks for One-Shot Image Recognition,” 2015.

[26] L. Bertinetto, J. Valmadre, J. F. Henriques, A. Vedaldi, and P. H. S. Torr, “Fully-Convolutional Siamese Networks for Object Tracking,” CoRR, vol. abs/1606.09549, 2016.

[27] A. Ussa, L. D. Vedova, V. R. Padala, D. Singla, J. Acharya, C. L. Zhang, G. Orchard, A. Basu, and B. Ramesh, “A low-power end-to-end hybrid neuromorphic framework for surveillance applications,” in British Machine Vision Conference - Workshops, 2019 (Accepted). Available: http://tinyurl.com/2aplgyt

[28] S. B. Shrestha and G. Orchard, “Slayer: Spike layer error reassignment in time,” in Advances in Neural Information Processing Systems, 2018, pp. 1412–1421.

[29] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2016, pp. 770–778.

[30] C. Brandli, R. Berner, M. Yang, S. C. Liu, and T. Delbruck, “A 240×180 130 db 3 μs latency global shutter spatiotemporal vision sensor,” IEEE Journal of Solid-State Circuits, vol. 49, no. 10, pp. 2333–2341, Oct 2014.

[31] P. A. Merolla, J. A. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura et al., “A million spiking-neuron integrated circuit with a scalable communication network and interface,” Science, vol. 345, no. 6197, pp. 668–673, 2014.

[32] J. Acharya, A. U. Caycedo, V. R. Padala, R. S. Sidhu, G. Orchard, B. Ramesh, and A. Basu, “EBBIOT: A Low-complexity Tracking Algorithm for Surveillance in IoT using Stationary Neuromorphic Vision Sensors,” in 32nd IEEE International System-on-chip Conference, 2019 (Accepted). Available: https://bit.ly/2S501DX

[33] B. Murmann and B. Hoefligner, NANO-CHIPS 2030, 1st ed. Springer International Publishing, 2020, pp. 191–192.

[34] Google. Edge TPU. [Online]. Available: https://cloud.google.com/edge-tpu

[35] L. Lin, B. Ramesh, and C. Xiang, “Biologically Inspired Composite Vision System for Multiple Depth-of-field Vehicle Tracking and Speed Detection,” in Asian Conference on Computer Vision, 2015, pp. 473–486.

[36] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in Advances in neural information processing systems, 2012, pp. 1097–1105.

Andres Ussa received his B.Sc. degree in Mechatronics Engineering from Nueva Granada Military University in 2012 and Joint M.Sc. in Embedded Computing Systems from TU Kaiserslautern and University of Southampton in 2016. His previous research experience has been focused on embedded systems design and machine learning applications. He had a short experience as a Software/Hardware Developer for consumer electronics.

Chockalingam Senthil Rajen received his B.Tech degree in Instrumentation and Control Engineering in 2020 and is currently pursuing his Masters in Mechanical and Aerospace engineering at the University of California San Diego. He had a short experience as a Machine Learning Engineer for AI services. His primary focus is on Robotics with a specialization in machine vision.

Tarun Puluri is currently pursuing his PhD at Drexel University. He received his B.Tech degree in Electronics and Communications Engineering from JNTUH College of Engineering Hyderabad and M.Engg degree in Microelectronics and Embedded Systems from Asian Institute of Technology in Dec’2018. He worked as a visiting researcher in NUS and later as an R&D Engineer at Panasonic R&D Center Singapore. His research interests include neuroscience, computer vision, and robotics.

Deepak Singla (M 20) received his B.Tech in Electrical Engineering (Power & Automation) from the Indian Institute of Technology, Delhi in 2018. After graduation, he joined Nanyang Technological University, Singapore as a Project Officer in the BRAIN Systems Lab - CICS headed by Dr. Basu. Since then, he is worked on IoT based applications with neuromorphic vision sensors. His research interests include neuromorphic engineering, brain-machine intelligence and computer vision.

Jyotibdha Acharya received her B.E. degree in ECE from Jadavpur University, India in 2014 and Ph.D. degree at HealthTech NTU, Interdisciplinary Graduate Program, Nanyang Technological University, Singapore. Previously, he worked in Philips Healthcare as a Graduate Engineer Trainee (Imaging Systems). His research interests include neuromorphic algorithms, deep learning, and biomedical applications.

Gideon Fu Chuanrong was an undergraduate student of the School of Computing Department at the National University of Singapore. His research focuses on Computer Vision, Image Processing, and Signal Processing. He pursued his research internship at the N.I. Institute for Health, NUS, Singapore during his undergraduate studies.

Arindam Basu received the B.Tech and M.Tech degrees in Electronics and Electrical Communication Engineering from the Indian Institute of Technology, Kharagpur in 2005, the M.S. degree in Mathematics and Ph.D. degree in Electrical Engineering from the Georgia Institute of Technology, Atlanta in 2009 and 2010 respectively. He is currently a Professor in City University of Hong Kong in the Department of Electrical Engineering.

Bharath Ramesh is currently a Senior Lecturer at International Center for Neuromorphic Systems, Western Sydney University. He works mainly in the research areas of pattern recognition and computer vision. At present, his research is centered on event-based cameras for autonomous sensing and navigation. He received the B.E. degree in electrical & electronics engineering from Anna University of Technology, Chennai, India in 2009; M.Sc. and Ph.D. degrees in electrical engineering from National University of Singapore in 2011 and 2015 respectively.