OBJECT DETECTION USING EVENT BASED CLUSTERING TECHNIQUE

C. Saraswathy M.E. [1], **V. Vijaykarthikeyan**[2]

Abstract

Clustering is crucial for many computer vision applications such as robust tracking, object detection and segmentation. Object detection/recognition finds its application in drones, autonomous driving, and so on. This work presents a real-time clustering technique that takes advantage of the unique properties of event-based vision sensors. Thus, this approach redefines the well-known clustering method using asynchronous events instead of conventional frames. Clustering accuracy reducing the computational cost by 88% compared to the frame-based method. The clustering achieved a consistent number of clusters along time. Event-based algorithm has been used for this cluster detection and we evaluated our method on a Verilog platform. This cluster detection mechanism is reliable, having less complexity on comparing with other image processing techniques.

KeyWords: Event-based processing, VHDL, Abnormal event detection, Linear feedback shift register(LFSR), cluster detection, Object Tracking, Cluster Object Tracker (COT).

DOI Number: 10.14704/nq.2022.20.8.NQ44701 **NeuroQuantology 2022; 20(8): 6770-6780**

[1] Associate Professor, ECE, K.S. Rangasamy college of Technology, Namakkal, India, saraswathy66@gmail.com
[2] PG Scholar, ME VLSI Design, K.S. Rangasamy college of Technology, Namakkal, India, vijaykarthikeyanv2020@srishakthi.ac.in

Introduction

With the increasing awareness of the public security, abnormal event detection (AED) in video surveillance plays a more and more important role in ensuring public safety. To achieve this, Image sensing and coding are essential. It makes sense to generate asynchronous output on an event-by-event basis. In recent years, great progress has been achieved using visual and inertial information. Visual Inertial Odometry (VIO) pipelines still struggle to cope with a number of situations, such as highspeed motions or high- dynamic range scenes. In this paper, Event-based filtering algorithms have been proposed to process the captured images. Driven by the need for effective feature processing on the event-stream, in this paper, a novel feature is proposed i.e., cluster object tracker, in a framework that runs asynchronously, considering only the sequential nature of the event-stream, in order to fully leverage the benefits of the event cameras regarding their high sensing rate. Moreover, the proposed approach is shown to be able to run in real-time even under high-speed event occurrence.

ABNORMAL EVENT DETECTION

A typical method to detect anomaly event is to detect patterns in video scenes that do not agree with the established normality. Existing AED methods [11], [12] can be mainly classified into three based categories: object-trajectory based methods, global-pattern based methods, and grid pattern- methods. Here, the first method is discussed. Object trajectory-based methods usually segment the crowd scene into different objects and then conduct objects tracking or identification. The trajectory of the object is the clue for abnormal events detection. The abnormality of objects' trajectories is often evaluated by zone-based analysis, fast matching algorithm, spatial-temporal path research, and deep learning algorithm. While existing methods show significant performance, they highly depend on advanced computer vision technologies such as accurate object detection and tracking for object trajectory-based methods, and robust feature descriptors extraction from images for global pattern- based methods. The heavy computation resources are always needed for image data processing. Further, when collecting data using standard frame-based cameras in static surveillance scenarios, it is unavoidable to record the redundant data from the background which are valueless for abnormal event detection. To address these issues, we propose a new abnormal event detection system.

EVENT PROCESSING ALGORITHM

Event-based processing refers to the processing of those information codified in events produced by event sensors. Since event-based sensors trigger events only when the intensity changes, the data is sparse, with low redundancy. However, in order to take advantage of event process the asynchronous event output to reduce sensor noise, extract low level features, and track objects, among others. These post- processing algorithms help to increase the performance and accuracy of further processing for tasks such as classification using spike-based learning, stereo vision, and visually-servoed robots, etc. In event cameras, whenever the intensity of an individual pixel varies beyond a specified threshold, an 'event' triggers in such pixel location and is reported asynchronously and independently from the rest of the pixels in the image array. Events are processed and often transformed into alternative representations that facilitate the extraction of meaningful information ("features") to solve a given task. This processing should preserve the low latency property of the sensor output, which comes about because of the asynchronous and quick readout of the sensors. Reference [3] presents the first event- based, energy efficient approach for object detection called PCA-RECT. Event cameras compress the visual scene in an asynchronous event-stream with high temporal resolution (in the order of μs) In addition to the elimination of any conventional time discretization (e.g., in the form of frames) in the perception pipeline, events only report incremental intensity changes at each pixel instead of the absolute intensity values (i.e., that conventional cameras capture in each frame). Consequentially, the Computer Vision community

is driven to revisit even the most established algorithms to process the event-stream. Specifically, they transmit per pixel intensity changes at the time they occur, in the form of a set of asynchronous events, where each event carries the space-time coordinates of the brightness change, and its sign. Event cameras transmit, in principle, all the information needed to reconstruct a full video stream and one could argue that an event camera alone is sufficient to perform state estimation. Event based computations transform real-world analog signals into asynchronous electrical event sequences. For that purpose, Event driven sensors are used as inputs for either algorithmic or hardware post- processing in event-driven neuromorphic systems which are gaining increasing interest in academia and industry for emulating the speed and power efficiency. The key question of the paradigm shift posed by event cameras is how to extract meaningful information from the event data to fulfil a given task.

Depending on how many events are processed simultaneously, two categories of algorithms can be distinguished: (i) methods that operate on an event-by- event basis, where the state of the system (the estimated unknowns) can change upon the arrival of a single event, thus achieving minimum latency, and (ii) methods that operate on groups or packets of events, which introduce some latency. Discounting latency considerations, methods based on groups (i.e., temporal windows) of events can still provide a detectable environment. The difference between both categories is more subtle: an event alone does not provide enough information for estimation, and so additional information, in the form of past events or extra knowledge, is needed. Orthogonally, depending on how events are processed, we can distinguish between model-based approaches and model-free (i.e., data-driven, machine learning) approaches. There are two main classes of event-based sensor processing algorithms: filters and feature extractors. Feature extractors extract a particular stimulus property such as edge orientation. Events can be transmitted with additional information such as the best feature at a pixel. Several of them arise from the need to aggregate the little information conveyed by individual events in the absence of additional knowledge. Event processing systems consist of several stages: pre-processing (input adaptation), core processing (feature extraction and analysis) and post-processing (output creation). The event representations may occur at different stages: for example, in an event packet is used at pre-processing, and motion-compensated event images are the internal representation at the core processing stage.

FEATURE DETECTION AND TRACKING

The two types of tracking are feature tracking and object tracking. Feature tracking finds its application in areas such as motion estimation, 3D reconstruction. Feature detection and tracking on the image plane are fundamental building blocks of many vision tasks such as visual odometry, object segmentation and scene understanding. Clustering aims at partitioning the space creating sets or clusters of elements that are as coherent as possible within the cluster and as different as possible from the elements in the other clusters. It is an unsupervised learning technique for which the cluster assignment or cluster number is unknown. The criterion that determines the classification given a specific element distribution is a distance measure on the space where cluster features are defined.

Mean shift clustering is widely used in segmentation and detection, tracking and optical flow estimation or feature matching for 3D reconstruction. Visual tracking aims at locating as accurately over time one or more targets (or clusters), in changing scenarios. Real-time visual tracking is a crucial task in Computer Vision, and still a challenge especially with clutter and multiple targets. Cameras capture and transmit a fixed number of frames regardless of the camera motion. The data may have motion blur, or large displacements and occlusions between consecutive frames, causing difficulties for clustering and tracking performance.

EVENT-BASED CLUSTERING USING MEAN SHIFT METHOD

In [4] the used method is the mean shift as

gradient descend for a robotic application in real-time; authors on [16] used a mixed alternative where the object is detected from a frame, and then it is followed in an event-based manner during the inter-frame time. Event-based sensors only transmit information when the intensity at a specific location change by a substantial amount; the sensor triggers an event e(x, t, p) where x is the location, t the time of the change, and p the polarity (positive or negative intensity change). To exploit the potential of the high temporal timing, we avoid accumulating events; each event is processed asynchronously when it happens. The mean-shift solution can be formulated as the gradient descent of the minimization in equation 2

$$arg \quad min \quad \mathbf{x}_i, \quad p_i, \quad f\delta(t_i)$$

(1)

$$\sum_{i,j} \quad K_G \ ([x_i \ , \ p_i \ , \ f_\delta(t_i)] - [x^{\ 0}{}_j \ , \ p_j \ , \ f_\delta(t_j \ )]/h)$$

(2)

An exponential decay function defined on the estimated lifetime of events that provides a good dynamic representation of local spatial structures. In the above equation $f\delta(t) = e - \Delta t / \tau$, where τ is a parameter that allows tuning the temporal history weight. Thus, we consider 4D data (space, polarity and time), so that the initial values are represented as [x 0 i, p0 i, fδ(t 0 i)] for pixel i. We use a multivariate Gaussian Kernel of the form $KG(x) = 1/\sigma \sqrt{2\pi}\ e^{-1/2(x\ T\ x)}$. Such a kernel results in better clustering although it usually requires more steps to converge. The minimization is defined over a summation of all pairs of pixels i, j. The tunnable bandwidth parameter h defines the kernel radius. In our case, at each iteration the current position is compared to the position of the original set x0 as in the classic method but the polarity and time function use the updated values of the previous iteration. Using this hybrid approach a better clustering is achieved while reducing the computational complexity. Classic mean-shift method is computationally expensive O(Kn2) where n is the number of pixels and K the number of iterations before convergence. Instead, this paper deals only with events that are processed in parallel in small packets of a few hundreds, helping to reduce the computational resource requirements. Clustering techniques are widely used for analysing the feature space in problems such as filtering and segmentation. However, most techniques require a-priori knowledge of the number or the shape of clusters and consequently, these techniques are not able to deal with real- world features. The mean shift technique computes for each point the mean of the data distribution (in some multi-dimensional parameter space) in a neighbourhood and then, the centre of this region is shifted to the computed mean until the processing converges. This algorithm is a nonparametric method that iteratively represents the feature space as a probability density function (PDF), where the modes of the density provide the denser regions and thus, the local maxima of the PDF. Therefore, the solution can be found as an iterative nonparametric density gradient estimation using a specific kernel for estimating the density function. No prior assumption on the number of clusters or their shape is made. Although the original Fukunaga's method works iteratively leading to better clustering results, Comaniciu's method is much faster reducing the number of comparisons in the feature space. We selected a hybrid approach that combines these two, solving the problem as a gradient descent for an optimization.

a. DATA STREAM FROM EVENT-BASED CAMERAS Let I(x, y, t) be the log-intensity value measured at the pixel location (x, y), where an event triggered at time t. A new event e is generated if, after an arbitrary period of time Δt, the absolute difference of I reaches a specific threshold K. Formally,

$\Delta I(x, y) = I(x, y, t + \Delta t) - I(x, y, t) = pK$, (3) where p denotes the event's polarity (i.e., is either

1 or −1) to indicate whether I increase or decreases. Ideally, a new event e = {t, x, y, p} is generated as soon as this condition is met, and appended asynchronously to the event-stream. In practice, however, the triggering of events in a single pixel is also subject to the sensor/hardware internal configuration (electronic biases). In fact, the accuracy of object detection and tracking relies on the large amount

of data for both online and offline model training.

b. CHALLENGES

Since events represent brightness changes, which depend on motion direction, one of the main challenges of feature detection and tracking with event cameras is overcoming the variation of scene appearance caused by such motion dependency. Tracking requires the establishment of correspondences between events (or features built from the events) at different times (i.e., data association), which is difficult due to the varying appearance. The second main challenge consists of dealing with sensor noise and possible event clutter caused by the camera motion.

c. TRACKING

For tracking, each incoming event was associated to the nearest existing blob/feature and used to asynchronously update its parameters (location, size, etc.). Circles and lines were treated as blobs in the Hough transform space. These methods were used in traffic monitoring and surveillance, high-speed robotic tracking and particle tracking in fluids or micro robotics. However, they worked only for a limited class of object shapes.

The temporal nature of these events is generally modelled with weights that are binary or an exponential decay function. Events modelled with binary weights can be represented with an activity buffer using built-in data structure support, first-in-first-out (FIFO). If a pixel's event exists for a longer duration than a predefined threshold 10 seconds, it is discarded. As this threshold impacts the bandwidth required for transmission, it must be picked with utmost care in view of the application requirement. In the case of decay function model, the event that occurred at time ti is made to lose its weight $e^{-ti/\tau}$ exponentially with time t, the decay being characterized by a time constant τ. The event is retained only if its weighted value surpasses a predefined threshold ϵ.

The background data is not recorded in event-based sensors, the images of non-stationary objects extracted from the event data represent salient regions, thus enhancing the accuracy of object recognition. The accompanying section brings out recent works on event-based object recognition. The methods discussed in this paper do not implement any techniques that compensate the background or camera motion, though there are few works available to compensate for ego-motion.

CLUSTER OBJECT TRACKER (COT)

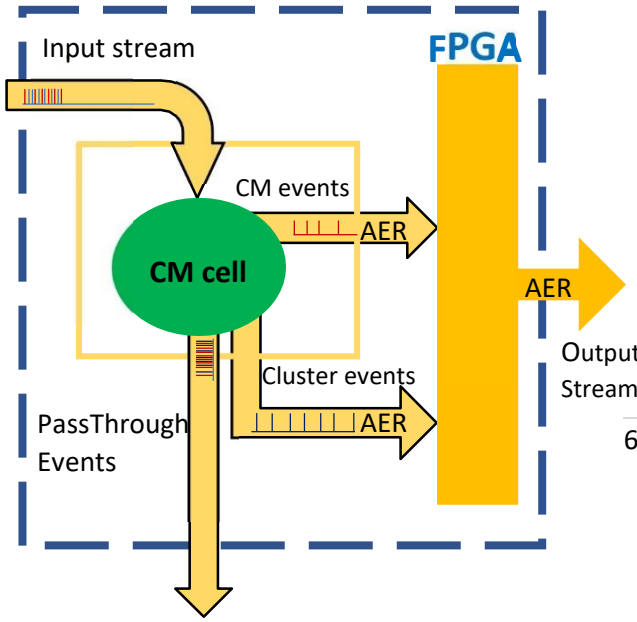

Fig 1. Architecture of Cluster Object Tracker

In post processing, each event indicates Object detection and tracking has encountered a high boost with the colossal development of deep learning algorithms. However, object recognition in event-based cameras still remains an open area of research. Over the decade, object tracking has advanced as an important application in domains such as robotics. The objects of intrigue, for the most part, include pedestrian, vehicles and so forth.

ALGORITHM

A COT has to detect potential objects from sensor output and then follows that object while it is moving through the visual field. It can be seen that a cluster (square part of the visual field) is detected when a configurable number of events are correlated both in time and space. To allow multiple objects detection and tracking, each tracker has to work with a reduced part of the visual field, which is called cluster. None of these

eISSN1303-5150

clusters can work with overlapped space addresses. When an event sensor senses a moving object, events sent are very close in time and their x and y addresses use to be close in space, because they belong to an object. If we focus the attention on a particular moving object, it is possible to filter all the activity not related to that object. Furthermore, this filtered activity can be mapped to the centre of a new reduced visual field that can be used as an input to a next processing layer in a more accurate way, like a ConvNet [13] classifier. The tracker can be modelled in a similar way. Nevertheless, in this work, a tracker is modelled as a feature extractor, where the centre of mass (CM) of the cluster activity is calculated and continuously sent out as a new stream of events. A tracker can be expressed mathematically as

$$E_{TRACKER} = E_{FC} \cup E_{CM} \quad (4)$$

where EFC is the filter cluster operation and ECM is the centre of mass operation. A cluster is understood as a squared part of the visual field around the tracked object. EFC and ECM are formally expressed as (5) and (6):

EFC = fFilterCluster(E) = {eFC0 , eFC1 , . . . , eFCs }, EFC ⊆ E, ∀eFCs = ei | (xCM − RC ≤ xi ≤ xCM + RC ;

yCM − RCM ≤ yi ≤ yCM + RC) (5)

ECM = fCentreMass(EFC)

= {eCM0 , eCM1 , . . . , eCMr }

∀eCMr | (xCMr = αxAVs + (1 − α)xCMr−1; yCMr = αyAVs + (1 − α)yCMr−1)

∀eAVs | (xAVs = αxFCs + (1 − α)xAVs−1; yAVs = αyFCs + (1 − α)yAVs−1) (6)

where EFC are the events from E that fall inside the cluster, (xAV , yAV) are the averaged centre of mass of last received events inside the cluster that are (xFC, yFC), ECM is a set of events that represent the smooth CM of the cluster over the history, (xCM , yCM) represents the current centre, RC is the cluster radius (half of a square side) and α is a mixing factor

SOFTWARE

The cluster object tracker algorithm is implemented as the Rectangular Cluster Tracker [14]. It is used in [15]. This algorithm processes event packets from an event sensor as follows:

1. For each event (of a packet), it finds a cluster that contains the event, based on a distance criterion. If a cluster exists, the cluster parameters (location and velocity) are updated using a mixing factor (α ≈ 0.01),

xn+1 = (1 − α)xn + αe (7)

where xn+1 is the updated location, xn is the old location and e is the current event.

2. If the last incoming event does not fall in any cluster, then a new cluster is inferred at this event location. This new cluster will be visible after it has received a configurable number of events (typically 5 events).

After all the events in a packet are processed according to steps 1 and 2, the algorithm processes all the clusters sequentially in the following way: (a) If a cluster does not receive any new event for a configurable time, this cluster is removed. (b) If two clusters have overlapping visual fields, they are merged into one new cluster. The new cluster location is computed by averaging the locations of the two clusters. This average is weighted according to the number of events accumulated by each cluster.

HARDWARE

This paper proposes a hardware COTS where each of the multiple trackers should is initialized to wait for an object at different initial locations and cluster sizes. As soon as a number of events, Nev, fall into the cluster within a configurable period of time, the object has been detected. A configurable extension over the cluster size is always monitored by the tracker for dynamic decision-making on cluster movements and cluster size updates. Nev can be adjusted dynamically for automatic adaptation to different object speeds and sizes as it is commented in following epigraphs.

EVENT COUNT CONSIDERATIONS

Since the event-based information corresponds to a dynamically moving visual stimulus, it is

necessary to compromise on the number of events (Nev) used for the CM calculation. If Nev is low, these events can vaguely represent the contour of the object, so the updated CM could be imprecise. In contrast, if Nev is large, then the current object location could be blurred. So Nev must be precisely adjusted for each particular scenario. Object speed also affects inversely the Nev parameter: If the object is moving slowly, it is preferable to collect more events to obtain a more precise CM calculation. But if the object movement is faster, then Nev must be decreased in order to have a good number of events that represent the current object location without blurring. In this paper we present a dynamic adjustment implementation for the Nev parameter in such a way that for each CM output, after processing Nev events, and taking into account the time difference between these Nev events for two consecutive CM calculations, Nev can be incremented or decremented.

LIFETIME MANAGEMENT

If a cluster, for any reason, stops receiving events from the sensor, it will be isolated in one region of the visual field. If the current detected object leaves the visual field of the sensor, or if the detected object was not a real object, the cluster will be isolated. To avoid this situation, a timer is able to reset the cluster tracker to its initial parameters. Every time the tracker's cluster receives an event, this timer is reset. If the timer overflows (after MAX clock cycles as commented in Table 1), the cluster tracker is reset to initial parameters.

COT FSM

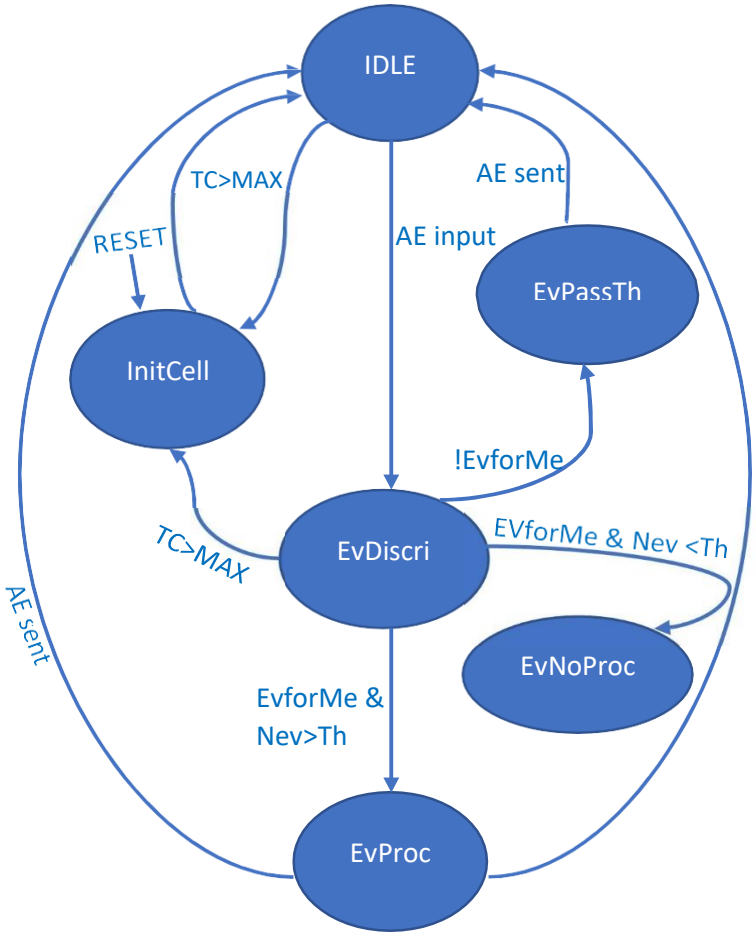

Fig 2. State diagram of COT FSM

Figure represents the simplified FSM diagram. InitCell state is the initial state after an asynchronous reset. In this state all the cluster parameters (shown in Table 1) are initialized with the values that come from the software interface. Then, the state machine gets to the idle state to wait for incoming events. In parallel, there are two timers running (TC and TC2). TC manages the reset of the cluster tracker because of the absence of incoming events. TC2 measures the needed time for collecting Nev events.

The previous needed time is compared to the current one to decide if Nev must be incremented or decremented dynamically by a factor of 2, as expressed in Fig. 2 with TH parameter. When a new event arrives, the state machine acknowledges and captures the address. Then it goes to the EvDiscri state in order to discriminate if the event falls inside the current cluster size or not. If the event does not fall in the cluster size, the state machine goes to the EvPassTh state where the event is sent out using the Pass Through port, and idle state is reached again to wait for next event. In contrast, if the received event fell in the cluster, then it depends on how many events have already been received in the cluster in order to perform the average calculation of the events in the cluster, or the calculation of the next centre of mass (CM). If current received event does not sum Nev events, then the FSM goes to EvNoProc state. In this state, the averaged X and Y addresses (Xi, Yi) for the last received events is updated with $\alpha = 1/2$ n as history (in order to use shift register operations instead of multiplications and divisions). Therefore, the averaged X and Y addresses are not taking into account all Nev events, but the state machine waits for them before performing the next calculations. When the current number of received events inside the cluster is exactly Nev, then the state machine moves to the EvProc state. In this state, g (the dynamic radius cluster) is updated taking into account the absence or presence of events in between the cluster radius and the internal sub cluster, as commented above. In this state it is possible to return to the InitCell state if the time since the last CM calculation was long (TC overflows). In the other case, the CM is calculated and the number of events (Nev) is updated according to those dynamic properties commented. Then an CM event is sent and the state machine will come back to the idle state.

TABLE 1. Cluster object tracker parameters

Symbol	Parameter Description
InitCMx	X axis of initial CM (centre cluster Y position)
InitCMy	Y axis of initial CM (centre cluster Y position)
InitRadix	Initial dynamic component of cluster radix (g) that corresponds to internal reduced cluster radix
RadixTH	Cluster radix is the addition of InitRadix plus RadixTH
RadixStep	Growing or shrinking steps of dynamic cluster radix component (g)
HistAvg	Number of past values to calculate an average before sending then next CM calculation. Needed in update(CM)
Max	Number of clock cycles without receiving events inside the cluster to be wait before resetting cluster tracker. Called *RSTimer* in the software interface.
ClustNev	Initial amount of events to be received for a CM calculation
RadixMin	Minimum allowed value for dynamic cluster radix (g)
RadixMax	*Max allowed value for* dynamic cluster radix (g)

SIMULATION SETUP

Simulation has been generated that mimics the box experiment described in the previous section. The architecture of the COT system is built upon the FSM model. The cluster is nothing but the aggregation of events stimulated. Clearly, events are defined as the pixel intensity changes occurs in the event camera. If those changes in the captured image are processed, they should be in the form of 1's and 0's bit stream. This could be even generated by the Verilog generating models which can seed random bit patterns. Those seeding can be implemented by LFSR.

1. LINEAR FEEDBACK SHIFT REGISTERS (LFSR)

LFSR stands for Linear Feedback Shift Register and it is a design that is useful inside of FPGAs. LFSRs are simple to synthesize, meaning that they take relatively few resources and can be run at very high clock rates inside of an FPGA. There are many applications that benefit from using an LFSR including:

1) Counters

2) Test Pattern Generators 3)Data Scrambling and 4)Cryptography

The linear feedback shift register is implemented as a series of D Flip-Flops inside of an FPGA that are wired together as a shift register. Several taps off of the shift register chain are used as inputs to either an XOR or XNOR gate. The output of this gate is then used as feedback to the beginning of the shift register chain, hence the Feedback in LFSR.

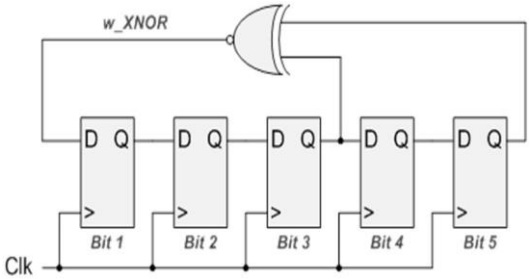

Fig 3. Block diagram of Linear Feedback shift Register

There are a few properties of shift registers that are important to note:

1. LFSR patterns are pseudo-random.

2. A pattern of all 0's cannot appear when the taps use XOR gates. Since 0 XORed with 0 will always produce 0, the LFSR will stop running.

3. A pattern of all 1's cannot appear when the taps use XNOR gates. Since 1 XNOR-ed with 1 will always produce 1, the LFSR will stop running.

Longer LFSRs will take longer to run through all iterations. The longest possible number of iterations for an LFSR of N-bits is 2N-

1. All possible patterns of something that is N- bits long is 2N. Therefore there is only one pattern that cannot be expressed using an LFSR. That pattern is all 0's when using XOR gates, or all 1's when using XNOR gates as the feedback gate.

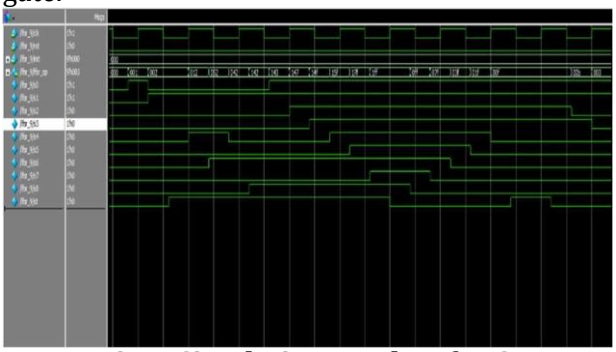

Fig 4. Simulation results of LFSR

The VHDL and Verilog code creates any N-Bit wide LFSR based on the desire. It uses polynomials (which is the math behind the LFSR) to create the maximum possible LFSR length for each bit width. Therefore, for 3 bits, it takes 23-1=7 clocks to run through all possible combinations, for 4 bits: 24-1=15, for 5 bits: 25-1=31, etc. I based this on an XNOR implementation to allow the FPGA to start up in an all-zero state on the LFSR. The LFSR uses D flipflop to generate the bit stream rather than the other flipflops. Later, it is XORed inside LFSR.

1. D FLIP-FLOP

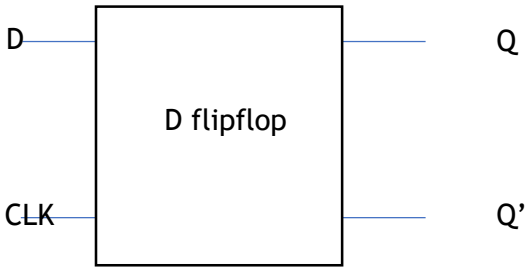

Fig 5. Block diagram of D flipflop

The above diagram shows the block diagram of simple D flipflop. The D flip-flop is a clocked flip-flop with a single digital input 'D'. Each time a D flip-flop is clocked, its output follows the state of 'D'. The D Flip Flop has only one input and the clock. The outputs follow the input D. Whenever the Reset is high, it follows the complement, otherwise it follows the input.

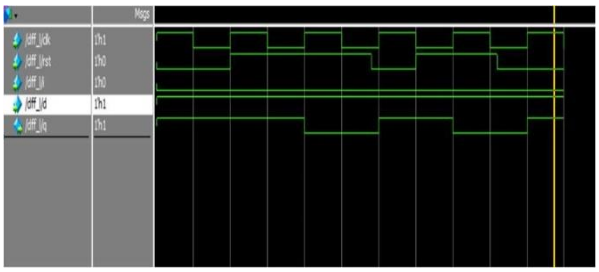

Fig 6. Simulation results of D flipflop

The LFSR as seen above simulation generates random bit pattern which is exactly looks like the real time event generated bit streams.

Table 2. COT FSM parameters for the experiment

Symbol	parameters	value
Nev	Initial number of events for CM calculation	8(slow) 3(fast)
TC	cluster inactivity reset time	20(slow) 7(fast)
TC2	counter to count the events	

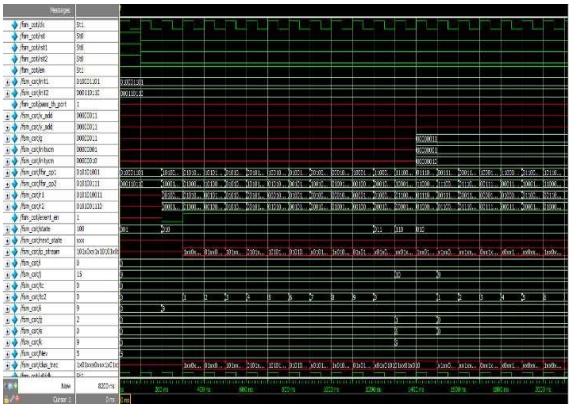

Fig 7. Simulation result of COT FSM

RESULTS AND DISCUSSION

The cluster object Tracker is implemented using Verilog HDL. The simulation is performed for the cluster tracker output under different situations. For, slow detection environment, Nev is set for 8 while for the faster one, the Nev is set for the count of 3 and 5. Because of the implementation of the state machines, there is no error in cluster event detection and in computations.

CONCLUSION

This paper presents the cluster object tracker mechanism and the proposed working model is successfully implemented. The design architecture is coded in Verilog platform. The proposed system is the indoor prototype model which can be further developed in real-time FPGA applications which needs expanded pre-processing stages. This COT model is compatible for the future applications like Deep learning, artificial Intelligence, etc.,

References

1. Alejandro Linares- Barranco, Fernando Perez- Peña, Diederik Paul Moeys, Francisco Gomez- Rodriguez, Gabriel Jimenez-Moreno, Shih-Chii Liu, Tobi Delbruck, "Low Latency Event-Based Filtering and Feature Extraction for Dynamic Vision Sensors in Real-Time FPGA Applications", September 13, 2019, Vol.7, pp. 134926-134942.
2. Stefano Soatto, Pietro Perona, "Reducing "Structure from Motion": A General Framework for Dynamic Vision Part 2: Implementation and Experimental Assessment", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 20, No. 9, September 1998, pp.943-960.
3. Bharath Ramesh, Andr´es Ussa, Luca Della Vedova, Hong Yang & Garrick Orchard, "PCA- RECT: An Energy-efficient Object Detection Approach for Event Cameras", April 2019.
4. Francisco Barranco, Cornelia Fermuller & Eduardo Ros, "Real-time clustering and multi- target tracking using event-based sensors", IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2018, pp. 5764- 5769.
5. Sangil Lee, H. Jin Kim, "Low-latency and Scene-robust Optical Flow Stream and Angular Velocity Estimation", Volume 4, 2016.
6. Antoni Rosinol Vidal, Henri Rebecq, Timo Horstschaefer, Davide Scaramuzza, "Ultimate SLAM? Combining Events, Images, and IMU for Robust Visual SLAM in HDR and High-Speed Scenarios", IEEE Robotics and Automation Letters, December, 2017.

7. Ignacio Alzugaray, Margarita Chli, "Asynchronous Corner Detection and Tracking for Event Cameras in Real-Time", IEEE Robotics and Automation Letters, June, 2018.
8. Linyan Cui, Chaowei Ma, "SDF-SLAM: Semantic Depth Filter SLAM for Dynamic Environments", May 14, 2020, Volume 8, pp. 95301-95311.
9. Annamalai Lakshmi, Anirban Chakraborty, Chetan
10. S. Thakur, "Neuromorphic vision: From sensors to event-based algorithms", Wiley Interdiscipl. Rev., Data Mining Knowl. Discovery, January 2019, vol. 9, no.4, p. e1310.
11. Minhao Yang, Shih-Chii Liu, Tobi Delbruck "A Dynamic Vision Sensor With 1% Temporal Contrast Sensitivity and In-Pixel Asynchronous Delta Modulator for Event Encoding", IEEE Journal of Solid-State Circuits, September 2015, Vol. 50, No. 9, pp. 2149-2160.
12. M. A. Sivilotti, ''Wiring considerations in analog VLSI systems, with application to field- programmable networks,'' California Inst. Technol., Pasadena, CA, USA, 1991.
13. K. A. Boahen, ''Communicating neuronal ensembles between neuromorphic chips,'' in Neuromorphic Systems Engineering. Boston, MA, USA: Springer, 1998, pp. 229–259.
14. C. Zamarreno-Ramos, A. Linares-Barranco, T. Serrano-Gotarredona, B. Linares-Barranco, ''Multicasting mesh AER: A scalable assembly approach for reconfigurable neuromorphic structured AER systems. Application to convNets,'' IEEE Trans. Biomed. Circuits Syst., vol. 7, no. 1, pp. 82–102, Feb. 2013.
15. T. Delbrück and P. Lichtsteiner, ''Fast sensory motor control based on event-based hybrid neuromorphic-procedural system,'' in Proc. IEEE Int. Symp. Circuits Syst. (ISCAS), May 2007, pp. 845–848.
16. M. Litzenberger, C. Posch, D. Bauer, A. Belbachir, P. Schon, B. Kohn, and H. Garn, ''Embedded vision system for real-time object tracking using an asynchronous transient vision sensor,'' in Proc. IEEE 12th Digit. Signal Process. Workshop, 4th IEEE Signal Process. Educ. Workshop, Sep. 2006, pp. 173–178.
17. D. Tedaldi, G. Gallego, E. Mueggler, and D. Scaramuzza, ''Feature detection and tracking with the dynamic and active-pixel vision sensor (DAVIS),'' in Proc. 2nd Int. Conf. Event-Based Control, Commun., Signal Process. (EBCCSP), Jun. 2016, pp. 1–7.