Event-based object detection and tracking - A traffic monitoring use case -

Simone Mentasti, Abednego Wamuhindo Kambale, and Matteo Matteucci

Department of Electronics Information and Bioengineering of Politecnico di Milano,
p.zza Leonardo da Vinci 32, Milan, Italy
{name.surname}@polimi.it

Abstract. Traffic monitoring is an important task in many scenarios, in urban roads to identify dangerous behavior and on-highway to check for vehicles moving in the wrong direction. This task is usually performed using conventional cameras but these sensors suffer from fast illumination changes, particularly at night, and extreme weather conditions. This paper proposes a solution for object detection and tracking using event-based cameras. This new technology presents many advantages to address traditional cameras limitations; the most evident are the high dynamic range and temporal resolution. However, due to the different nature of the provided data, solutions need to be implemented to process them in an efficient way. In this work, we propose two solutions for object detection, one based on standard geometrical approaches and one using a deep learning framework. We also release a novel dataset for this task, and present a complete application for road monitoring using event cameras. [1]

Keywords: Event-cameras · Object detection and tracking · Road monitoring.

1 Introduction

Event cameras, also known as neuromorphic cameras, are a new type of sensor inspired by the working process of the human retina. Unlike traditional cameras, they do not acquire a complete frame at a constant speed. Instead, each sensor pixel works independently and returns a value representing the brightness change only when it occurs for that specific pixel. This new type of sensor possess several advantages over traditional frame cameras; they have a high temporal resolution in the µs range, high pixel bandwidth, low power consumption, and high dynamic range [1]. These properties enable them to give an high throughput in different applications such as pose estimation [2], Simultaneous Localisation And Mapping (SLAM) [3], autonomous robots perception, and object tracking [4]. In particular, event-based object tracking algorithms can be divided into two classes: the event-driven tracking method and the pseudo-frame-based tracking method. The first one involves processing each event without accumulating

[1] Dataset available at: https://airlab.deib.polimi.it/datasets-and-tools/

it in a frame and requires ad-hoc solutions designed especially for these types of sensors. The second accumulates events into a binary pseudo-frames that can be used as input to more traditional computer vision algorithms.

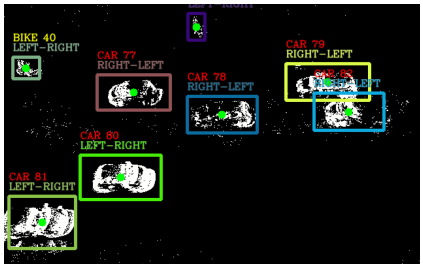


Fig. 1: Example image of the working system in a two lane scenario, for each object the system computes a 2D bounding box, assigns an ID, and performs tracking displaying the moving direction

In this work, we propose a pseudo-frame-based detection and tracking approach using a Prophesee third-generation event-based camera [5]. The goal of our project is to develop a robust system for object detection and tracking in outdoor scenarios, with a particular interest in road monitoring, as shown in Figure 1. Compared to a traditional camera, event-based sensors provide different advantages in this specific task. In particular, the high dynamic range allowed us to develop a solution that works equally well during night and day, without the risks of having the sensor dazzled by traffic lights. Similarly, due to the high frame rate, rainy conditions are not an issue since the drop of rain will be just some tiny dots on the image, which can be easily filtered.

This work investigates two different approaches to the problem: a model-free algorithm that uses only geometrical features to detect objects and a YOLOv4-based [6] approach which uses a deep learning model to detect cars, bikes, and pedestrians. The final component of the pipeline is the tracking module [6], which uses a Kalman filter [7] and the Hungarian algorithm [8] for detection, association, and ID assignment. The proposed solution is a robust pipeline that can track moving objects in the camera's field of view. In particular, the system has been designed for road monitoring. It can track cars, bikes, and pedestrians and monitors a set of useful parameters like the object's direction and the number of objects by category. Moreover, it can raise alarms in case of vehicles moving in the wrong direction. When the system is mounted on the highway, it is a handy feature to identify vehicles entering ramps in the opposite direction. This work aims to demonstrate the effectiveness of event cameras in the detection and tracking context, their suitability for deep learning, and provide the community with an annotated dataset to train such architectures.

This paper is structured as follows. In Section 2 we present the state of the art on object detection and event-based sensors, although, due to the novelty of the

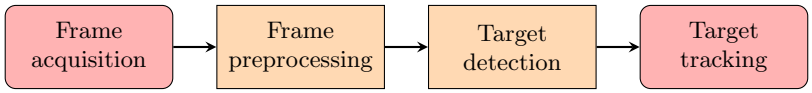


Fig. 2: Frame-based object tracking pipeline.

sensor, the literature is fairly limited. Then in Section 3 we propose our pipeline for object detection and tracking from event data. For the detection component, two different solutions are proposed. Finally, in Section 4 we present some numerical results to highlight the advantages of the best performing solution, and Section 5 draws some conclusions and highlights future development.

2 State of the art

Traditionally, frame-based object detection and tracking consist of four steps, as shown in Figure 2. First, the *Frame acquisition process* outputs a frame that feeds the *preprocessing* stage, in charge of removing noise and keeping relevant information for further steps.

The *target detection* phase consists of separating the object's pixels from the unwanted signal. The detection can be performed either without a machine learning based object detector or with the help of one. In the first case, a common approach is to use nearest neighbor algorithm [9], [10] to perform Connected Component Labeling (CCL) [11], clustering groups of pixels to form objects. Then performing Connected Component Analysis (CCA) [12],[13] to determine the area, the centroid, and the bounding box of every detection. Finally it is possible to use these features to categorize different objects in the scene [14], [15]. This can be done using geometrical solutions, but also classical machine learning models like Support Vector Machine [16] and AdaBoost [17]. A second approach is to use deep learning (DL) methods to perform object detection [18].

Most DL-based object detection models treat object detection as a regression/classification problem and use end-to-end architectures built on regression/classification to map pixels to bounding box locations together with class probabilities. This operation has the advantage of reducing computational time. One significant and representative framework is You Only Look Once (YOLO) [19]. The YOLO framework predicts both bounding boxes and confidences for multiple categories. The original architecture works with 416x416 images at 45 fps in real-time using consumer GPUs. A simpler variant called Fast YOLO [20] can process images at 155 fps with a slight loss in accuracy. YOLOv2 is an enhanced version that was later proposed in [21], which presents some improvements to the original architecture, like batch normalization, multi-scale training, and anchor boxes.

The final stage of the *association and target tracking* aim to connect single frame detections to a common target and create a track that captures target's current position and prior movements. The goal is to find an optimal matching

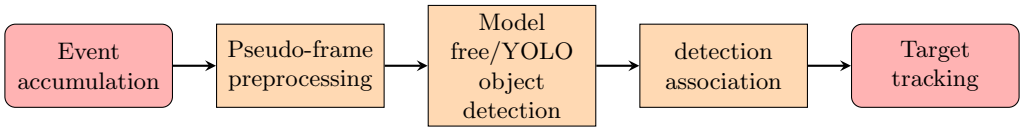


Fig. 3: Schema of the object detection pipeline.

between the tracked objects and these predictions. A widely used algorithm for this task is the Hungarian algorithm [22].

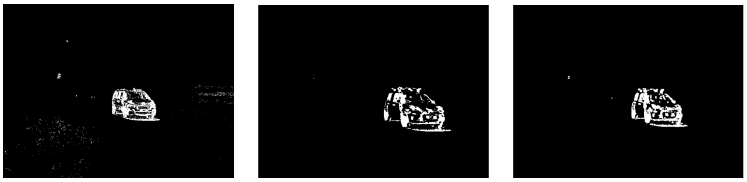
While all the presented techniques were designed to work on traditional images, proper adaptation is required to process event-based streams. The literature identifies two distinct ways to achieve target detection and tracking for event-based cameras [23]. One approach uses pseudo-frames, whereas the other performs direct detection on filtered events. The corresponding techniques for frame-based sensors previously highlighted differ slightly for event-based detection when using pseudo-frames. The process starts naturally by capturing the raw event data; undesirable events are filtered to extract meaningful object motion, and frames are built. Then traditional techniques can be employed to process the reconstructed image. The solutions based directly on events, process instead each event asynchronously and independently. Two different methods have been used in the literature for this second approach, one using adaptive time surfaces [24] and another using events association in 3D Point [25], [26]. The main advantage of these approaches is that the event-based sensor's high temporal resolution is fully utilized, still, a custom solution needs to be developed, and it is impossible to adapt from standard computer-vision approaches.

3 Object detection and tracking from event data

This paper proposes a pipeline for object detection and tracking from event data acquired by a Porphesee event-based camera. In particular, we present two alternative approaches; the first one, which we call "Model-free", is based on geometrical operations on the pseudo-frame to retrieve the objects. The second instead leverages a convolutional neural network to extract a list of targets. Nevertheless, the initial processing phase and the clusters tracking are common between the two solutions, as shown in Figure 3.

3.1 Preprocessing

Pseudo-frames are event-based versions of conventional video frames created by dividing continuous event data into evenly spaced time bins, or frames, to create an image. They are similar to standard binary frames except that they are sparse, with areas with no events set as black. This similarity with binary images enables traditional detection and tracking algorithms to use pseudo-frames as their input with minimal changes.



(a) Reconstructed input pseudo-frame. (b) Pseudo-frame after erosion. (c) Pseudo-frame after dilation.

Fig. 4: Evolution of the pseudo-frame through the preprocessing pipeline.

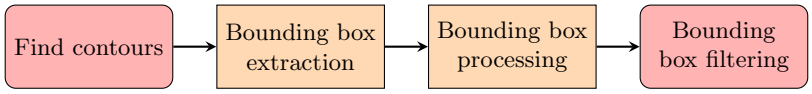


Fig. 5: Schema of the "model-free" pipeline.

The accumulation of events into pseudo-frames starts by receiving the stream of events from the event-based sensors. Each event from the received stream is added into a queue used to generate frames depending on the user-defined event accumulation time (for our test, we employed a window of $20ms$, $dt = 20ms$, which guarantees a frame rate of $50Hz$). The generation of a frame at a specific time T takes all the events whose timestamp t lies between $T - dt$ and T.

Due to the sensor's high sensitivity, not all received events are valid data; some are just noise. Therefore, the preprocessing step helps filtering the data and discards noisy points that may negatively affect detection. Moreover, an object represented by the events is not always continuous; some object regions may appear without events affecting the object detection phase. For this reason, the last step of the preprocessing pipeline is a sequence of morphological operations, as shown in Figure 4. In particular, the best results were achieved by applying first an erosion filter to remove single noisy points from the sensor, which are fairly frequent in this type of camera, (Figure 4a). Then we apply a dilation filter to combine clusters close together, and therefore belonging to the same obstacle. This preprocessing phase is fundamental, particularly for the "Model-free" obstacle detector, since this is based only on geometrical constraints and separate components can negatively affect system performance.

3.2 Model-free detection

The "Model-free" detector relies on the frames provided by the pseudo-frame filtering process. Figure 5 shows the pipeline used for this approach. Starting from the filtered pseudo-frame, the first operation finds the objects in the scene by detecting the contours using the algorithm presented in [27]. From the obtained contours, we extract the associated rectangular bounding boxes by retrieving the rectangle's location, width, and height that best fit each contour area. A final step of bounding box processing is required to group bounding boxes near each

other which are more likely to represent the same object. Also, in this last step, we discard bounding boxes that lie inside others. The latter is a fairly common issue since event cameras generally return obstacles' contours and color discontinuity on an obstacle can generate a smaller target inside the obstacle bounding box (e.g., car windows).

3.3 Yolo-based detector

A YOLOv4 model has been applied to provide a more reliable and accurate detection than the "Model-free" pipeline, but it requires a training dataset and more computational power. Unlike the "Model-free" pipeline, our YOLOv4 model has been designed to identify different object categories from the pseudo-frame. However, accurate detection results require a well-trained model, which depends on the availability of a good dataset. Therefore, we built our dataset from multiple recordings of real-world scenarios to train YOLOv4 and YOLOv4-tiny models. We used a dataset consisting of 3.1k images, which has been obtained manually annotating 1255 images and then performing data augmentation using the Roboflow tool [28]. Data autmentation techniques included flipping, rotation, translation and cropping. We were interested only in detecting cars, bikes, and pedestrians for this task, so these are the only label used in the dataset. We used transfer learning to speed up the training phase, starting from the YOLOv4 pre-trained weights provided online. We used Recall, Precision, and F1-score to evaluate the performance of our YOLOv4 model. In addition, the mean of Average Precision (mAP) metric is used to compute the average of interpolated Precision to Recall for each detected object (see Section 4).

3.4 Object tracking

The object detection phase computes the bounding boxes locations associated with the targets of interest in the visual scene. Then the object tracking phase associates these detections to the trackers in each video frame. A good tracking algorithm should assign one and only one ID to each object in the visual scene. For each new object entering the scene, a new tracker should be assigned independent from the other objects already in the scene.

For each new detection, we assign a predictor whose role is to model the object's movement in the scene. Then, each predictor is associated with a tracker. In this work, we used Kalman filters as predictors. A Kalman filter tracks a single object by modeling its movement. The filter holds the target state x with information on its center coordinate (x, y), its width-height ratio, its area, the variation of its area over time, and its velocity. The observation z holds only observable information from a single frame, i.e., the center coordinates, the aspect ration and the area. Furthermore, the filter keeps some additional data related to the object, such as the time the object has been visible, and for how long the position has been predicted due to absence of measurement.

The detection association step aims to determine the best possible fit between predictors and detections. First,we generate a cost matrix whose element

c_{ij} denotes the similarity (or dissimilarity) between a predictor i and a detection j. We used the Intersection over Union (IoU) between two bounding boxes to compute similarity. We also combined other similarity measures to compute the cost, such as shape and distance measures. The shape and distance features consider the geometry and the position of the bounding box. We adopted the approach proposed in [29] to compute the similarity measures between two bounding boxes, and these similarity measures are multiplied to give advantages to the bounding boxes that have similarities both in position and shape instead of considering just one feature. These terms are the Euclidian distance of the shape and the position as from Equation (1), (2) and (3), where A represents the detection and B the tracked object. In particular Equation (2) computes the similarity value on the size, therefore H and W indicates the two bounding box dimensions. Equation (3) computes the position similarity, with x and y as the coordinates of the bounding box centroid.

$$C(A, B) = c_{shape}(A, B) \times c_{dist}(A, B) \tag{1}$$

$$c_{shape}(A, B) = e^{-w_2 \times (\frac{|H_A - H_B|}{H_A + H_B} + \frac{|W_A - W_B|}{W_A + W_B})} \tag{2}$$

$$c_{dist}(A, B) = e^{-w_1 \times ((\frac{x_A - x_B}{W_A})^2 + (\frac{y_A - y_B}{H_A})^2)} \tag{3}$$

In our case, we set $w_1 = 0.5$ and $w_2 = 1.5$. Obtained the cost matrix, we then use the Hungarian algorithm to assign a detection to a predictor and update it. We choose the Hungarian algorithm [22] because it gives a solution to the assignment problem in polynomial time.

We use a different tracking engine for each object category to achieve precise object tracking when objects categories are mixed in the same environment. Thus, when an object is detected as a car, bike, or pedestrian, it is passed directly to the corresponding tracking engine. The framework developed provides the moving object's direction and counts the different objects considering their categories. To extract the object direction, we use its associated predictor as it models the object's movement on the image plane. Indeed, the state x_k contains the vertical component of the velocity v_y and the horizontal component v_x.

To retrieve the object's direction, we compute the norm V of the velocity from v_x and v_y and the angle between v_x and v. First we perform a check on the norm, if it is outside a reasonable threshold it is discarded, and we assume a wrong data association was performed. Otherwise in the case the *angle between* v_x *and* $V <= 60°$, we return $LEFT - RIGHT$ if $v_x > 0$ or $RIGHT - LEFT$ if $v_x < 0$. If the *angle* $> 60°$ we return $UP - DOWN$ if $v_y > 0$ or $DOWN - UP$ if $v_y < 0$.

We also allowed the user to select areas to specify the correct moving direction between two points. An alarm is raised when an object is detected moving in the opposite direction. For example, Figure 6 presents the interface for two opposite direction lanes. The top one in green moves from left to right, the bottom one in purple from right to left.

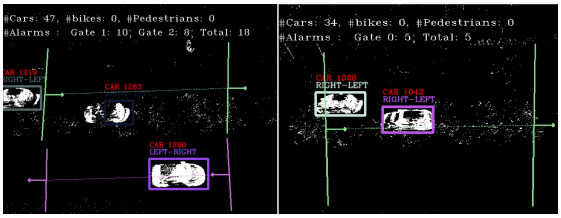


Fig. 6: Example image of the working system in a two lane scenario (left) and on a single lane scenario (right).

4 Experimental results

The two proposed approaches have been validated using a dataset acquired with the event-based camera (third-generation Prophesee event-based camera [5]). The experimental setup did not employ a GPU to have a lightweight system, but both the "Model-free" and the Yolov4 have been run on an i7-9750H, 2.60GHz CPU.

The model-free approach presented above has as its main advantage a tailorable frame rate that can reach 300 fps, on the testing hardware, including also the processing time (i.e., pseudo-frame filtering, bounding box extraction, bounding box filtering). This is a considerably high value for tasks like road monitoring. However, this approach presents some significant limitations. Since the only features used to detect and classify an object are the bounding box's width, height, and area, it is challenging to differentiate multiple object categories accurately. Also, this approach is not particularly robust in very noisy scenarios, and it has many parameters to fine-tune to obtain promising results. Nevertheless, it can run at high frequency on any low-power device, thanks to its simple rule-based approach.

In the case of the YOLOv4 approach, the model training has been performed using a TK80 GPU on Google Colab. For the training phase, we employed the following hyper-parameters: a batch size of 64, with 12 subdivisions that denote the number of pieces the batch is broken into for GPU memory, and a maximum number of batches of 6000. The network was trained to detect only three classes of interest, pedestrians, bicycles, and cars. The best achieved results for the model during the validation with mAP0.5 (mean Average Precision, with Intersection over Union threshold for the bounding boxes of 0.5) was 0.88. For this model, on the testing set, we had a value of 0.83. Table 1 provides the AP (Average Precision) per class for the YOLOV4 model. As it can be noticed, we achieved a good AP for each class.

Our YOLOv4 based detection model is also robust when facing a noisy environment since only objects are detected, and all the noise is discarded. The only disadvantage of this solution is that the YoloV4 model is relatively complex and drastically limits the system framerate. In particular, it achieved only 4 fps on the CPU i7-9750H, 2.60GHz, while reaching 16 fps on the Google Colab GPU

Class	**Validation** (%)	**Testing** (%)
Car	97.3	98.55
Bike	88.69	87.58
Pedestrian	66.98	62.51

Table 1: YOLOv4 validation and testing AP value per class.

Class	**Validation** (%)	**Testing** (%)
Car	98.46	94.82
Bike	90.82	44.59
Pedestrian	75.70	48.7

Table 2: YOLOv4-tiny validation and testing AP value per class.

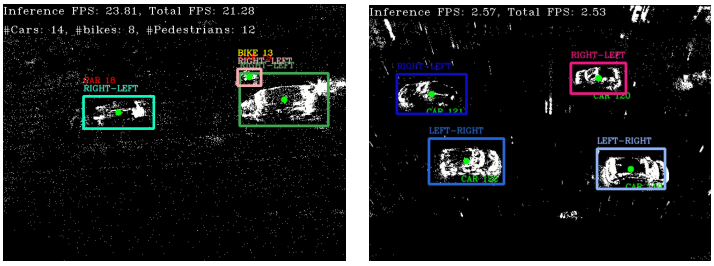
	P	**R**	**F1-score**	**IOU** (%)
Validation	0.85	0.85	0.85	66.63
Testing	0.68	0.72	0.7	47.94

Table 3: Detection metrics for confidence threshold of 0.25.

TK80. To address this issue, we trained a new model on a smaller network, which would lead to less accurate predictions but shows an increased maximum framerate. In particular, we tested the YOLOv4-tiny model. The main difference between YOLOv4-tiny and YOLOv4 is a significant reduction in the network size, making this model considerably faster and suited even for CPU-only tasks. The training process performed is similar to the one of YOLOv4. The global performances are shown in the Table 2.The achieved results are worse than the Yolov4 model, particularly for small objects, but are still acceptable. In particular, if we consider that a Kalman Filter-based tracking is performed after the detection phase. Moreover, this solution is faster, even when working only on the CPU.

Considering our application we decided a confidence threshold of 0.25 to be enough to achieve good results in tracking. Table 3 provides the model Precision, Recall, and F1-score and the average IoU considering the confidence threshold of 0.25. The values are always above 0.5, which indicates the potential of this solution. In particular, not as a standalone one, since the detector might miss some object, but as an accurate source for the Kalman Filter to track the moving target in the scene.

It was impossible to evaluate our model against other available models since the literature related to object detection from event-based cameras is extremely poor, and no available model could be found while performing this test. Instead, a significant comparison can be performed between the two trained solutions, YOLOv4-tiny and Yolov4. While the latter was slow and therefore discarded for this task, Yolov4-tiny could achieve an inference frame rate of 37 fps on CPU i7-9750H, 2.60GHz, and 76 fps on the TK80 Google Colab GPU, making it suitable



(a) Detection in a night scenario.

(b) Detection in a rainy scenario.

Fig. 7: Example images of the YoloV4-tiny detections in some challenging scenarios.

for our object detection task. Moreover, the drop in accuracy between the two approaches was not significant.

To conclude, we compared this solution against the original "model-free" approach. The deep-learning approach shows its robustness in noisy scenarios like rain, night (with the light from the vehicle), and camera movement, as shown in Figure 7. Moreover, compared to the model-free scenario, the detection confidence threshold is the only hyper-parameter to consider while using this model. The limitation of this approach is the frame rate since the model inference time constitutes the bottleneck, but moving the inference to a GPU-based system can be considered a viable solution if needed. Finally, it is particularly interesting to notice how the proposed solution, based on an event-based camera, can perform extremely well in this scenario where traditional cameras would struggle due to weather and lighting conditions.

5 Conclusions

To conclude, in this paper, we presented two solutions for object detection and tracking from event-based data. Events acquired by the camera are accumulated using a binary pseudo-frame that can be processed using two different algorithms. The model-free approach fits well scenarios without a lot of noise and can run at high speed. Therefore it offers the possibility to adjust the frame rate depending on application constraints. The model-free approach is also suitable for low-power systems; nevertheless, it performs poorly with excessive noise and challenging weather conditions. Contrary, the two YOLOv4 models offer better performance and can be used in very noisy scenarios too. But, they are resource-eager, and that leads to a power-consuming system. Moreover, they work at best on systems that use GPUs since, on CPU, they offer a relatively low frame rate. However, the modified version, the YOLOv4-tiny model, can run at an acceptable framerate on a system equipped with a good CPU but no GPU, like the testing hardware. The tracking module has proven its ability

to perform the intra-frame data association correctly. In such a way, we could compute each object moving direction and identify vehicles moving in the wrong direction. Moreover, using a Kalman Filter to predict the obstacle's position is extremely useful for consistently providing an object position, even when the detector misses the object for a single frame.

References

1. Guillermo Gallego, Tobi Delbruck, Garrick Orchard, Chiara Bartolozzi, Brian Taba, Andrea Censi, Stefan Leutenegger, Andrew Davison, Joerg Conradt, Kostas Daniilidis, and Davide Scaramuzza. Event-based vision: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 4 2019.
2. Elias Mueggler, Henri Rebecq, Guillermo Gallego, Tobi Delbruck, and Davide Scaramuzza. The event-camera dataset and simulator: Event-based data for pose estimation, visual odometry, and slam. *The International Journal of Robotics Research*, 36(2):142–149, 2017.
3. David Weikersdorfer, David B Adrian, Daniel Cremers, and Jörg Conradt. Event-based 3d slam with a depth-augmented dynamic vision sensor. In *2014 IEEE international conference on robotics and automation (ICRA)*, pages 359–364. IEEE, 2014.
4. Marco Cannici, Marco Ciccone, Andrea Romanoni, and Matteo Matteucci. Asynchronous convolutional networks for object detection in neuromorphic cameras. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 0–0, 2019.
5. Prophesee. prophesee.ai website, 2021.
6. Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*, 2020.
7. Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. 1960.
8. G Ayorkor Mills-Tettey, Anthony Stentz, and M Bernardine Dias. The dynamic hungarian algorithm for the assignment problem with changing costs. *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-07-27*, 2007.
9. James P Boettiger. A comparative evaluation of the detection and tracking capability between novel event-based and conventional frame-based sensors. 2020.
10. Vandana Padala, Arindam Basu, and Garrick Orchard. A noise filtering algorithm for event-based asynchronous change detection image sensors on truenorth and its implementation on truenorth. *Frontiers in neuroscience*, 12:118, 2018.
11. Federico Bolelli, Michele Cancilla, Lorenzo Baraldi, and Costantino Grana. Toward reliable experiments on the performance of connected components labeling algorithms. *Journal of Real-Time Image Processing*, 17(2):229–244, 2020.
12. Linda G Shapiro. Connected component labeling and adjacency graph construction. In *Machine intelligence and pattern recognition*, volume 19, pages 1–30. Elsevier, 1996.
13. Florian Lemaitre, Arthur Hennequin, and Lionel Lacassagne. Taming voting algorithms on gpus for an efficient connected component analysis algorithm. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7903–7907. IEEE, 2021.

14. Lifeng He, Xiwei Ren, Qihang Gao, Xiao Zhao, Bin Yao, and Yuyan Chao. The connected-component labeling problem: A review of state-of-the-art algorithms. *Pattern Recognition*, 70:25–43, 2017.
15. Federico Bolelli, Stefano Allegretti, Lorenzo Baraldi, and Costantino Grana. Spaghetti labeling: Directed acyclic graphs for block-based connected components labeling. *IEEE Transactions on Image Processing*, 29:1999–2012, 2019.
16. Bernhard Scholkopf. Support vector machines: a practical consequence of learning theory. *IEEE Intelligent systems*, 13, 1998.
17. Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.
18. Zhong-Qiu Zhao, Peng Zheng, Shou tao Xu, and Xindong Wu. Object detection with deep learning: A review. *IEEE Transactions on Neural Networks and Learning Systems*, 30:3212–3232, 7 2018.
19. Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.
20. Mohammad Javad Shafiee, Brendan Chywl, Francis Li, and Alexander Wong. Fast yolo: A fast you only look once system for real-time embedded object detection in video. *arXiv preprint arXiv:1709.05943*, 2017.
21. Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017-January:6517–6525, 12 2016.
22. Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.
23. Anton Mitrokhin, Cornelia Fermüller, Chethan Parameshwara, and Yiannis Aloimonos. Event-based moving object detection and tracking. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–9. IEEE, 2018.
24. Christian P Brändli. *Event-based machine vision*. PhD thesis, ETH Zurich, 2015.
25. Samya Bagchi and Tat-Jun Chin. Event-based star tracking via multiresolution progressive hough transforms. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 2143–2152, 2020.
26. Marco Cannici, Marco Ciccone, Andrea Romanoni, and Matteo Matteucci. A differentiable recurrent surface for asynchronous event-based data. In *European Conference on Computer Vision*, pages 136–152. Springer, 2020.
27. Satoshi Suzuki et al. Topological structural analysis of digitized binary images by border following. *Computer vision, graphics, and image processing*, 30(1):32–46, 1985.
28. Roboflow. roboflow website, 2021.
29. Ricardo Sanchez-Matilla, Fabio Poiesi, and Andrea Cavallaro. Online multi-target tracking with strong and weak detections. In *European Conference on Computer Vision*, pages 84–99. Springer, 2016.