POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Executive Summary of the Thesis

EVENT-BASED CAMERA FOR OBJECT TRACKING

Laurea Magistrale in Computer Sciences and Engineering - Ingegneria Informatica

Author: ABEDNEGO WAMUHINDO KAMBALE

Advisor: Prof. Matteo Matteucci

Co-advisor: Simone Mentasti

Academic year: 2021-2022

1. Introduction

Event cameras 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 a reasonable throughput in different applications such as pose estimation and Simultaneous Localisation And Mapping (SLAM), autonomous robots perception, and object tracking.

In this thesis we focused on the later. Event-based object tracking generally uses two methods: the event-driven tracking method and the pseudo-frame based tracking method. The first method involves processing each event without accumulating it in a frame and requires new approaches than the traditional computer vision approaches. The second accumulates events into pseudo-frames that can be used as input to traditional computer vision algorithms.

This work proposes a pseudo-frame based tracking approach using the Prophesee third-generation event-based camera. Since the event-based camera is sensitive to noise, the proposed approach had to be robust in different scenarios likes rain, ground motion, night and camera motion. This work investigates two methods: a model-free approach that uses the geometrical features to detect objects and a YOLOv4-based approach which uses a trained model to detect detect cars, bikes, and pedestrians. The tracking module uses Kalman filtering and the Hungarian algorithm for detection association and ID assignment. The final product is a robust C++ monitoring framework to track moving objects in the camera's field of view. The system monitors a part of a road, it tracks cars, bikes, and pedestrians and monitors some parameters like the object's direction, the number of objects by category and the alarms.

2. Literature review

Researchers have investigated different approaches to offer good tracking performance considering applications constraints. They can be divided into two main groups: the frame-based and the event-based tracking approaches.

2.1. Frame-based detection and tracking

Frame-based object detection and tracking consists of four steps as shown in Figure 1. The frame capturing process outputs a frame that feeds the pre-processing stage which is in charge of discarding noises and keeping relevant infor-

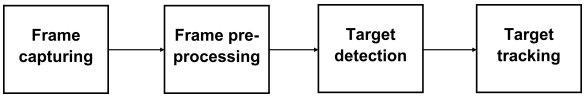

Figure 1: Frame-based object tracking pipeline

mation to use in further stages.
The target detection consists of an algorithm that accurately separates an object's pixel from an unwanted signal. The detection is done either without a model or with a trained model. In the first case some works propose a nearest neighbour methods to cluster pixels to form a object. The algorithm determines the area, the centroid, and the bounding box of every single detection. One can consider some features to categorize different objects in the visual scene.
Some approaches use the classical machine learning models like Support Vector Machine and AdaBoost and other methods use the deep neural network to achieve the object recognition. The deep learning (DL) methods differ from the traditional ones in terms of network architecture, optimisation techniques, and training approaches [6].
Some DL-based object detection models treat object detection as a regression/classification problem and use a unified framework to obtain final results. Theses frameworks are one-step architectures built on global regression/classification and map pixels directly to bounding box locations with the class probabilities. This operation has the advantage to reduce time. One significant and representative framework is You Only Look Once (YOLO).
The YOLO framework predicts both bounding boxes and confidences for multiple categories. The YOLO architecture can handle 416x416 images at 45 fps in real-time, while a simpler variant called Fast YOLO can process images at 155 fps with better results than existing real-time detectors. YOLOv2 is an enhanced version that was later proposed in [3], which uses many interesting techniques like batch normalization, multi-scale training and anchor boxes.
The detection association and target tracking steps aim to connect the stand-alone detections to a common target and create a track that captures that target's current position and prior movements. The goal is to find an optimal matching between the tracked objects and the predictions. Some algorithms, like the Hungarian algorithm [2], were proposed to solve the assignment problem.

2.2. Event-based detection and tracking

There are fundamentally two distinct ways to achieve target detection and tracking for event-based cameras. One approach uses pseudo-frames, whereas the other performs directly detection on filtered events.
The corresponding techniques for frame-based sensors outlined earlier in Section 2.1, 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.
Approaches based on events process events asynchronously and independently. There are two different methods that have been used in the literature one using adaptive time surfaces and another using events association in 3D Point. The main advantage of these approaches is that the event-based sensor's high temporal resolution is fully utilized.

3. Our proposed approach

Figure 2 presents the pipeline of the approach we adopted for this work. The following subsections describes each stage in the algorithm.

3.1. Pseudo-frames events accumulation

Pseudo-frames are event-based versions of conventional video frames created by dividing continuous event data into evenly spaced time bins, or frames. They are similar to standard frames except that they are sparse, and this similarity enables standard detection and tracking algorithms to use them as their inputs.
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. This accumulation time helps accumulate frames for a longer period dt since the number of generated events at a specific time T can be limited (considering the events' microsecond resolution). The generation

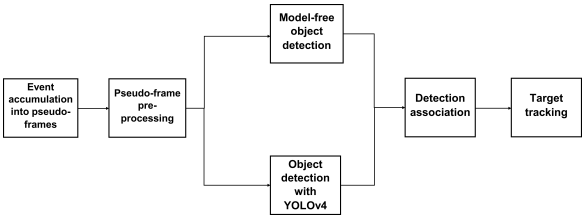

Figure 2: Pipeline of the adopted methodology

of a frame at a specific time T takes all the events whose timestamps t lie between $T - dt$ and T.

3.2. Pseudo-frame pre-processing

All the accumulated events do not represent the targets of interest, some do, and others are noise. The preprocessing step helps discarding noise that may affect the detection process negatively. It also helps increase the significance of the target of interest in the visual scene. An object represented by the events is not always continuous; some object regions may appear without events affecting the object detection phase. This work used four morphological filters for the pseudo-frame preprocessing: erosion, dilation, opening, and closing.

3.3. Object detection

This stage constitutes the backbone of the tracking algorithm. A tracking exists if there is an object detected, and false detections imply lousy tracking. This fact implies that the frames supplied to the detection process should be free from noise, or the detection process should discard the noise.

3.3.1 Model-free object detection

Model-free detection relies on the frames provided by the pseudo-frame filtering process. Figure 3 shows the pipeline we 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 in [5].
From the obtained contours, we extract the associated bounding boxes by extracting the location, the width and height of the rectangle that best fit each contour area. The bounding box processing helps grouping some bounding

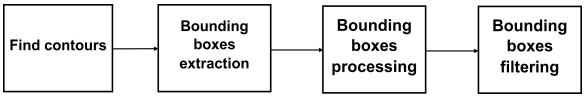

Figure 3: Model-free object detection pipeline

boxes that appear to be near each other and are more likely to represent the same object. Also, it helps discarding the bounding boxes that lie inside others.
The last process in the model-free detection pipeline is bounding box filtering. This step is more application related since the bounding box size (width, length) or the bounding box area depend on the application field and the system setup (position of the sensor). In this stage, we filter bounding boxes depending on the geometric dimensions since all the available bounding boxes do not necessarily represent an object.

3.3.2 Object detection with YOLOv4

The YOLOv4 model helps identify different object categories in a visual scene. Good detection results rely on a well-trained model, which depends on the availability of a good dataset. We made our dataset from the recording we took from real-world scenarios to train YOLOv4 and YOLOv4-tiny models. We used a dataset consisting of 3,1k images. The considered label are cars, bikes and pedestrians. We used transfer learning, starting from the YOLOv4 pre-trained weights provided online.
We used Recall, Precision and F1-score to evaluate the performance of the YOLOv4 model. In addition the mean of Average Precision (mAP) metric is used computing the average of interpolated Precision to Recall for each detected object.
The formula in Equation (1) show how to calculate the interpolated Precision.

$$P_{interp}(R) = \max_{\tilde{R}>R} P(\tilde{R}) \tag{1}$$

Then, for each class, the average of the interpolated Precision on Recall to obtain the AP.

$$AP = \frac{1}{N} \sum_{R \in \{0, 0.1, \ldots, 0.9, 1\}} P_{interp}(R) \tag{2}$$

Finally, the mAP is computed from the average AP considering all classes as in equation (3).

$$mAP = \frac{1}{c}\sum_{i=1}^{C} AP_i \quad (3)$$

3.4. Object tracking

The object detection phase provides the bounding boxes locations associated with the targets of interest in the visual scene, while 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 a new object entering the scene, a new tracker should be assigned independently on the other objects already in the scene.

We adopted the approach of assigning a predictor to a detection, whose role is to model the object's movement in the visual scene. Then, each predictor is associated to a tracker. In this work, we used Kalman filters as predictors. A Kalman filter tracks a single object by modelling its movement. The filter holds the current x and the predicted state $\hat{x}$. The state x has data related to the bounding box, namely its centre coordinate (x, y), its width-height ratio, its area, the variation of its area over time and its velocity. On the other hand, the observation z holds only information that is observable from a single frame. Furthermore, the filter holds some additional data related to the object, the time the object has been visible and for how long the predictor has been in an inactive state.

The purpose of the detection association step is to determine the best possible fit between predictors and detections. First,we generated the cost matrix whose element c_{ij} denotes the similarity (or dissimilarity) between a predictor i and a detection j. We used the IoU between two bounding box to compute this matrix elements. We also combined similarity measures to compute the cost. These similarity measures are shape and distance measures. The shape and distance features consider the geometry and the position of the bounding box.

We adopted the approach in [4] to compute the similarity measures between two bounding boxes, and similarity measures are multiplied to give advantages to the bounding boxes that have similarity both in position and shape instead of considering just one feature. These terms are the Euclidian distance of the shape and the position as seen in Equation (4), (5) and (6).

$$C(A, B) = c_{shape}(A, B) \times c_{dist}(A, B) \quad (4)$$

$$c_{shape}(A, B) = e^{-w_2 \times (\frac{|H_A - H_B|}{H_A + H_B} + \frac{|W_A - W_B|}{W_A + W_B})} \quad (5)$$

$$c_{dist}(A, B) = e^{-w_1 \times ((\frac{x_A - x_B}{W_A})^2 + (\frac{y_A - y_B}{H_A})^2)} \quad (6)$$

In equation (6) A is the detection and B the tracked object. In our case, we set $w_1 = 0.5$ and $w_2 = 1.5$.

Obtained the cost matrix, we use the Hungarian algorithm to assign a detection to a predictor and update it. We choose the Hungarian algorithm [2] because it gives a solution to the assignment problem in polynomial time.

3.4.1 Multi-object and multi-categories tracking

To achieve precise object tracking when objects categories are mixed in the same environment we adopted a tracking engine approach for each object category; 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 direction of the moving object and count the different objects considering their categories. To extract the object direction we use its associated predictor as it models the movement of object. 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. If $V < threshold\_V$ *or* $V > V\_max$ the function return 0 and the object is discarded. Otherwise in the case the *angle between* v_x *and* $V <= 60°$, it returns $LEFT - RIGHT$ if $v_x > 0$ or $RIGHT - LEFT$ if $v_x < 0$. If the $angle > 60°$ it returns $UP - DOWN$ if $v_y > 0$ or $DOWN - UP$ if $v_y < 0$.

We provided also an option to draw polygons surrounding noisy areas and save these polygons into a configuration file that the tracking system loads before it starts working. In this way, we reduce the noisy areas and enable the tracking process to achieve better results.

4. Results and discussion

This section discusses the results of the implemented tracking framework. We used a CPU

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.

	mAP0.5 (%)	mAP0.25 (%)
Training	73.87	-
Validation	66.39	85
Testing	62.7	68

Table 2: Training, validation and testing mAP.

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

Table 3: 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 4: Detection metrics for confidence threshold of 0.25.

i7-9750H, 2.60GHz and the TK80 Google Colab GPU to evaluate the models.

4.1. Model-free approach

We implemented the model-free approach presented above as it has the advantage to offer a tailorable frame rate that can reach 300 fps on the testing hardware considering the processing time (i.e., pseudo-frame filtering, bounding box extraction, bounding box filtering), which is a good value for many applications.
However, we observed some limitations of this approach. The only features used to detect and consider an object are the bounding box's width, height, and area, making it difficult to differentiate different object categories. Also this approach is not robust in very noisy scenarios and it has many parameters to control to obtain promising results.

4.2. YOLOv4-based approach

Google Colab helped perform the YOLOv4 training using the TK80 GPU. We chose the following hyper-parameters for YOLOv4 model configuration: a batch size of 64, 12 subdivisions which denote the number of pieces the batch is broken into for GPU memory, maximum number of batches set to 6000. We used 3 classes.
In the end, the achieved validation mAP0.5 was 0.88, while for the stand-alone testing set, we had 0.83. Table 1 provides the AP per class for the YOLOV4 model. We have a good AP for each class.
The detection model is also robust when facing a noisy environment, only objects are detected, and all the noise is discarded. The only problem with this model was the frame rate limitation. We achieved 4 fps on a CPU i7-9750H, 2.60GHz, and reached 16 fps on the Google Colab GPU TK80.
The low frame rate led us to choose another model with lower accuracy than the YOLOv4, which helps achieve a higher frame rate, the YOLOv4-tiny. The main difference between YOLOv4 tiny and YOLOv4 is the reduction in the network size.
The YOLOv4-tiny training process is similar to the one of YOLOv4. The global performances are shown in the Table 2.
The value mAP0.5 denotes the mean average precision considering a confidence threshold value of 0.5, while mAP0.25 is for a confidence threshold value of 0.25. Table 3 shows how the model performs for each class and help understand in which way one should to improve the model. The model performs well for cars and should be tailored for bikes and pedestrians. The reason for that could be the size of the bikes and the pedestrian in the visual scene. One approach that can be used to mitigate this problem is to move the camera near the objects. In our application, a confidence threshold of 0.25 is enough to achieve good results. Table 4 provides the model Precision, Recall and F1-score and the average IoU considering the confidence threshold of 0.25.
At this point, it is not possible to evaluate our model against other available models since this work is the first one, to our knowledge, to

deal with the event-based pseudo-frames using YOLOv4-tiny for car bikes and pedestrians detection.
An interesting result obtained by YOLOv4-tiny compared to Yolov4 is the inference frame rate achieved that can reach 37 fps on CPU i7-9750H, 2.60GHz, and 76 fps on the TK80 Google Colab GPU.
YOLOv4-tiny model-based object detection offers several advantages compared to the model-free approach. It shows its robustness in noisy scenarios like rain, night (with the light from the vehicle), and camera movement. Also, compared to the model-free scenario, the only hyperparameter to consider while using this model is the detection confidence threshold. The limitation of this approach is the frame rate since the model inference time constitutes the bottleneck.

5. Conclusion

After implementing and testing the presented tracking algorithms, we come up with these conclusions. The model-free approach fits well for scenario without lot of noises. It offers the possibility to adjust the frame rate depending on application constraints. The model-free approach is also very adapted to low power systems.
The YOLOv4 model offers better performance and can be used in very noisy scenario and it will still be robust. However it is resource-consuming and that lead to power-consuming system. It is adapted to system using GPUs since on CPU it offers a relatively low frame rate. However the modified version, YOLOv4-tiny model, fits well also for some systems without many resources for instance the testing hardware that achieved 37fps was a CPU i7-9750-H, 2.6GHz.
The information about the object direction can have an advantage while investigating the traffic violation on the road or help plan the pose of a moving robot depending on the direction of a moving obstacle. The object counting can be used in predicting the traffic on the road for good management of the traffic in smart cities.
The futures developments of our works may include to developing an event-driven approach that considers grouping asynchronous events into clusters, and using some feature descriptors to identify objects. Also one can investigate the use of simple feature descriptors with the model-free approach to categorize objects in the visual scene since simple feature descriptors would keep the high frame rate. The extension of the dataset can also be a plus to consider different camera setups to make the framework usable in different scenarios.

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] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 3 1955.

[3] 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.

[4] Ricardo Sanchez-Matilla, Fabio Poiesi, and Andrea Cavallaro. Online multi-target tracking with strong and weak detections. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9914 LNCS:84–99, 2016.

[5] Satoshi Suzuki and Keiichi A. be. Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing*, 30:32–46, 4 1985.

[6] 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.