ISSN 2007-9737

Video Object Tracking by Feature Point Descriptor and Template Matching

Andrés E. Pat-Chan[1,*], Francisco J. Hernandez-Lopez[2], Mario R. Moreno-Sabido[1]

[1] Tecnológico Nacional de México/IT de Mérida, Mexico

[2] SECIHTI-Centro de Investigación en Matemáticas A.C., Mexico

{mg13080859, mario.ms}@merida.tecnm.mx, fcoj23@cimat.mx

Abstract. The present research focuses on developing a method based on feature point descriptors and template matching and comparing its performance with a method based on deep learning. These methods have particular aspects in how they were implemented; some stand out for the simplicity of their structure and others for the complexity they entail. The methods presented in this work range from developing a basic template matching algorithm, developing an algorithm based on feature point descriptors incorporating the template matching qualities to obtain better results, to implementing a method based on deep learning. Performance and precision tests are carried out to compare the methods on a selected dataset of video object tracking.

Keywords. Video object tracking, template matching, feature point descriptors, deep learning.

1 Introduction

Object detection and tracking applications have been growing in various sectors, such as video surveillance, transportation, agriculture, industry, sports, health, and academia. The process of estimating over time the location or bounding box of one or more objects through a video is called video tracking or video object tracking [10]. Various visual resources, videos, images, and video tracking methods currently fit multiple projects and applications [15, 8].

The traditional methods to perform the video tracking are those based on optical flow, template matching, background subtraction or change detection, feature point descriptors, and correlation filters [14, 1, 13]. Among the methods based on deep learning for object detection are models based on convolutional neural networks, such as R-CNN and YOLO [7], which, combined with methods such as intersection over union, the Hungarian algorithm, and the Kalman filter, can also track these detected objects [6].

The methods selected for this work meet specific qualities in their structure to obtain relevant results that can be compared appropriately. These methods range from template matching through feature point descriptors to implementing deep learning. Having an essential element such as the video and candidate objects to track, the different methods provide helpful information that, although in some cases they are synthesized and easy to analyze, in others they are not, so the problem lies in carrying out a study on these video object tracking methods and subject them to evaluations that demonstrate their robustness and reliability.

This paper aims to conduct a study where various video object tracking methods are implemented and evaluated to obtain relevant and helpful information in projects or applications with visual resources. With information from tests and experiments, it is possible that video object tracking projects can save research and testing time and have a solid foundation for developing applications that use video object tracking [11]. On the other hand, currently, the growth of tools that contribute

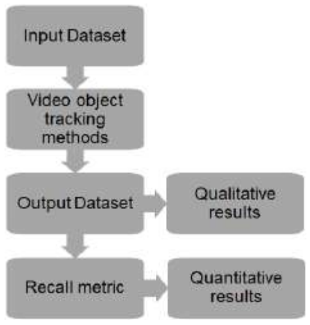

Fig. 1. Proposed methodology

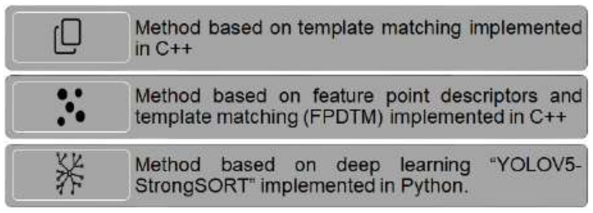
Method based on template matching implemented in C++
Method based on feature point descriptors and template matching (FPDTM) implemented in C++
Method based on deep learning "YOLOV5-StrongSORT" implemented in Python.

Fig. 2. Video object tracking methods implemented

to the tracking of objects in video has grown exponentially, and that is why developers need information and data that they can use to make decisions and carry out their projects successfully.

This paper is organized as follows. Section 2 presents the proposal methodology, describing the dataset, tracking methods, and metrics used to evaluate the implemented methods. The experimental results are presented in Section 3, where qualitative and quantitative results are shown. Section 4 presents the conclusions and future work.

2 Methodology

This research focuses on the implementation and analysis of the different algorithms and methods for tracking objects in video, ranging from the most basic algorithms or approaches to the most complex, viewing it from a technical point of view. The proposed methodology is designed to achieve the previously stated objectives, not ruling out discoveries. In Fig. 1, a block diagram of the methodology is observed. Next, the steps of the proposed methodology and the computational resources used are described in detail.

2.1 Input Dataset

The dataset we used for this research was obtained from the VOT Challenge website [2]; this site contains various freely licensed image sequences used to evaluate object tracking algorithms [18]. It is called an image sequence since in each folder, there is a series of image files; each image sequence has a certain number of image files, in addition to preserving the uniformity of the size of the image files contained in each folder or sequence of pictures. The image sizes in the sequences range from 320 x 240 pixels to 640 x 480 pixels. Table 1 shows the relationship of the image sequences downloaded and used in this research.

Furthermore, the dataset includes the ground-truth ($\texttt{GT\_BBoxes}$), which is a text file that stores in each line the coordinates of a bounding box; this must have the following format: (x, y, w, h), where (x, y) is the upper left coordinate, and (w, h) is the width and height of the bounding box, respectively.

2.2 Video Object Tracking Methods

Below are the methods implemented in this work (see Fig. 2). Some methods are implemented in C++, while others are in Python. Likewise, an image as an icon on the left makes a general illustrative representation of the central strategy used by the algorithm or method.

2.2.1 Template Matching

The template matching (TM) method consists of taking a template image and finding it in a larger image. When the algorithm finds the image region most similar to the template, it is highlighted to show where the most similar region is located. This algorithm is very useful for finding specific objects in images and is the basis for developing several computer vision projects [5].

Fig. 3 exemplifies the template matching method in a general way. Given the bounding box (BBox) in the first frame (f_1) of the input video, a subregion or template of the image is obtained. With the

Table 1. Image sequences of input dataset

ID	Sequence	N_f	Brief Description
1	Ball	603	Ball in motion.
2	Board	698	Electronic circuit board in motion.
3	Box	1161	Small cardboard box in motion.
4	Car	374	Car leaving a parking lot.
5	Car_2	945	White van on streets.
6	Carchase	9928	Police chase.
7	Cup_on_table	1021	A cup on a table with the camera moving.
8	Dog1	1390	A moving dog plush.
9	Gym	767	Olympic gymnast presenting her routine.
10	Juice	404	Juice box on a table with the camera moving.
11	Jumping	313	A person jumping a rope.
12	Lemming	1336	Teddy bear in motion.
13	Liquor	1741	Bottles in motion.
14	Mountain-bike	228	Motorcyclist jumping a ramp.
15	Person	948	A person in motion.
16	Person_crossing	1018	A person walking through a park.
17	Person_p_occluded	306	A person filmed from different angles with partial occlusion.
18	Singer	351	A singer presenting his show.
19	Sylvester	1345	A Sylvester plush in motion.
20	Track_running	503	Professional runner in a competition.

template T established, the searching process follows, where T is compared with all possible image subregions in a sliding window strategy. The sliding window is applied over a searching area instead of the whole image to reduce the computational processing.

To compare T and the subregions in the searching area, the normalized cross-correlation (NCC) measure can be used, which is defined as follows:

$$\texttt{NCC} = \frac{\sum_{i=1}^{M}(T_i - \bar{T})(f_{i,w} - \bar{f}_w)}{\sqrt{\sum_{i=1}^{M}(T_i - \bar{T})^2}\sqrt{\sum_{i=1}^{M}(f_{i,w} - \bar{f}_w)^2}}, \quad (1)$$

where i denotes a pixel in the image, M is the number of pixels in T, w is a window of size equal to T centered in a pixel in the searching area, $\bar{f}_w$ is the average of the pixel values in the window w, and $\bar{T}$ is the average of the pixels values in the template T. The NCC can take values between $[-1, 1]$, while closer to one, the T and the subregion f_w are more similar. See [13] for more details.

2.2.2 Feature Point Descriptor and Template Matching

Inspired by the ALIEN tracker based on feature point descriptors [14], we developed the Feature Point Descriptors and Template Matching (FPDTM) tracker.

Figure 4 shows the general flowchart of the FPDTM method. Given the BBox in the first frame (f_1) of the input video, the features and descriptors are computed using the SIFT algorithm to initialize the object and context feature sets T_1 and C_1, respectively. The state $\vec{x}_1$ includes the object center location (x_t, y_t), scale s_t and rotation angle θ_t with respect to the BBox. Furthermore, we use a template vector T_V, consisting of a set of ten

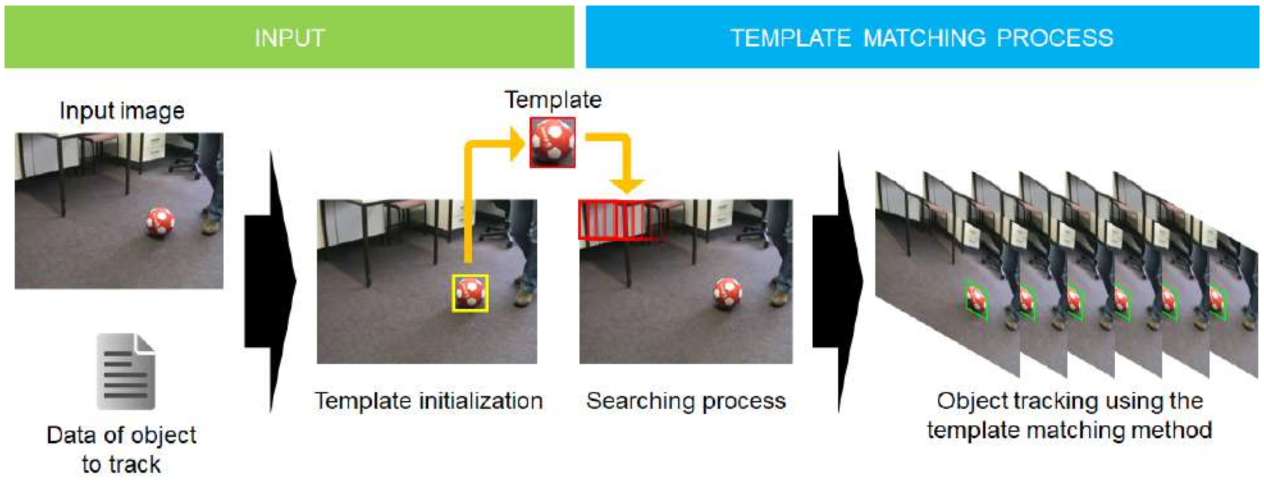

Fig. 3. General process of the template matching method

templates or subregions corresponding to different object views. The initialization of the template vector corresponds to loading the object to track in the first position of the vector.

For the following frame f_t, a search region is computed to avoid searching for the object in the whole frame. Features and corresponding descriptors are computed at frame f_t. A procedure to find the matching between the object and context feature sets of frames f_{t-1} and f_t is performed, filtering the features corresponding to the object feature set that does not match the context feature set. Then, the object state $\hat{x}_t$ is estimated using the filtered object feature set, minimizing the loss function reported in [14].

With the estimated object state $\hat{x}_t$, we compute the matching between the image obtained with the BBox at $\hat{x}_t$ and the template vector images using the NCC measure. We obtain the maximum NCC (`NCC_max`). When the template vector is complete, we must compare ten NCC values to obtain the `NCC_max`.

If the scale, rotation, and NCC meet the following conditions: $(|\hat{s}_t - \hat{s}_{t-1}| < k_s)$, $(|\hat{\theta}_t - \hat{\theta}_{t-1}| < k_\theta)$, and $(\texttt{NCC\_max} > \texttt{thr\_NCC})$, then the object is detected. If the object is detected, an occlusion detection procedure is performed; otherwise, the process continues with the following frame.

If the cardinality of the context feature set that contains the features localizing inside the BBox area at state $\hat{x}_t$ is higher than the threshold N_O, then the object is occluded. If the object is occluded, the following step is to update the template vector; otherwise, the object and context appearance are updated previously to continue with the update of the template vector.

If the object is not occluded, the object and context feature sets are updated, including new features and descriptors. When the cardinality of some set exceeds a threshold, some features and descriptors are removed in a uniform random sampling way; see [14] for more details.

Suppose the `NCC_max` is found in a location l different from zero of the template vector. In that case, the image in this location is updated using a linear combination as follows:

$$T_V^{t+1}(l) = \alpha f_t + (1-\alpha) T_V^t(l), \tag{2}$$

with $\alpha \in [0,1]$ a parameter which controls the contribution of f_t and $T_V^t(l)$. The corresponding procedures are repeated until the last frame of the video is reached.

2.2.3 YOLOV5-StrongSORT

There is a yolo_tracking framework available in [3, 4], where multi-object trackers are implemented

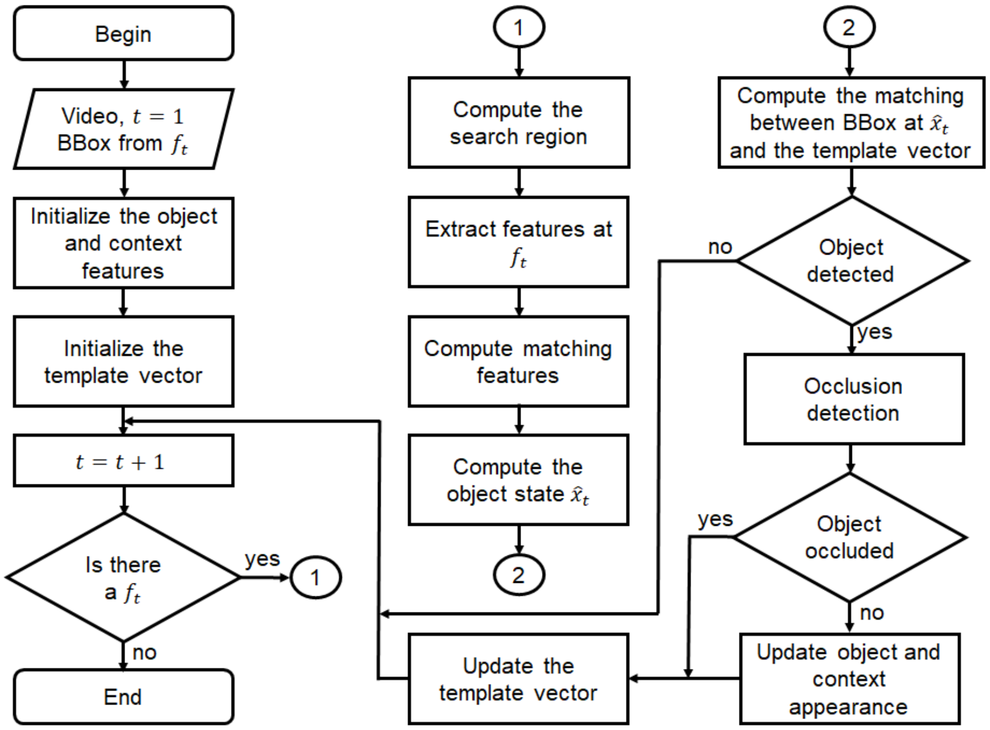
Begin
Video, $t = 1$
BBox from f_t
Initialize the object and context features
Initialize the template vector
$t = t + 1$
Is there a f_t
yes
1
no
End
1
Compute the search region
Extract features at f_t
Compute matching features
Compute the object state $\hat{x}_t$
2
Update the template vector
2
Compute the matching between BBox at $\hat{x}_t$ and the template vector
Object detected
no
yes
Occlusion detection
Object occluded
yes
no
Update object and context appearance

Fig. 4. General flowchart of the FPDTM method

in Python. We tested the YOLOV5-StrongSORT method, which uses the YOLOV5 object detector model inside the StrongSORT tracker method. The YOLOV5 model is open source, actively maintained by Ultralytics [16] and commonly trained in the COCO dataset [9]. StrongSORT [6] is a tracker method based on DeepSORT [17], which consists of an automatic object detector and a combination of the Kalman filter and Hungarian algorithm for tracking the detected objects. The Kalman filter predicts the object movement from a linear velocity model. On the other hand, the Hungarian algorithm is used to solve an assignment cost matrix, which is constructed from the intersection over the union between each detection and all predicted bounding boxes of the targets.

Different from DeepSORT, StrongSORT is equipped with advanced modules such as a detector model based on YOLO, a more robust appearance feature extractor, an exponential moving average to update the embedding features, a correlation coefficient maximization for camera motion compensation, and a matching cost that combines the appearance and motion information [6]. Thus, the detections generated by YOLOV5 are passed to StrongSORT, which combines appearance and motion information to track the objects trained in the YOLOV5.

2.3 Output Dataset

The output dataset consists of text files (`Out_BBoxes`) with the same format as the GT_BBoxes generated by the object tracking methods. With the Out_BBoxes, it is possible to analyze the results qualitatively and quantitatively.

2.4 Recall Metric

This part involves the text files `GT_BBoxes` and `Out_BBoxes`, which are used to calculate the recall metric to analyze the effectiveness of each method.

Before calculating the recall metric, the intersection over union ($\mathtt{IoU}$) between the $\mathtt{GT\_BBoxes}$ and $\mathtt{Out\_BBoxes}$ is computed as follows:

$$\mathtt{IoU}_i = \frac{A(\mathtt{GT\_BBoxes}_i \cap \mathtt{Out\_BBoxes}_i)}{A(\mathtt{GT\_BBoxes}_i \cup \mathtt{Out\_BBoxes}_i)}, \quad (3)$$

where A denotes the area and i indicates a register in both set. Thus, for each frame i of a video sequence, the $\mathtt{IoU}_i$ evaluates the ratio between the intersection area and union area of the ground-truth bounding box $\mathtt{GT\_BBoxes}_i$ and the estimated bounding box $\mathtt{Out\_BBoxes}_i$ [12].

The recall metric is defined as follows:

$$r = \frac{\mathtt{TP}}{\mathtt{TP} + \mathtt{FN}}, \quad (4)$$

where $\mathtt{TP}$ represents the true positives and $\mathtt{FN}$ the false negatives, which are computed considering a threshold $\tau \in (0, 1)$ over the $\mathtt{IoU}_i$. Thus, if the value $\mathtt{IoU}_i > \tau$, the estimated bounding box $\mathtt{Out\_BBoxes}_i$ is taken as a $\mathtt{TP}$; else, it is taken as $\mathtt{FN}$.

3 Experiments and Results

Two programming languages were used: C++ and Python. These languages easily fit this research work, and many scientific projects are implemented in these two languages. OpenCV library was used because it has several methods that help develop image and video processing tasks. Furthermore, Matlab was used for the analysis of quantitative results. In terms of hardware, a PC with an 8-Core i7 CPU and 16 GB RAM was used locally. In addition, a server with Intel(R) Core(TM) CPU i9-9920X 3.50 GHz, Ubuntu 20.04 (64-bit), 24 hyper-threading cores, and 64 GB RAM was used remotely.

In the following subsections, the qualitative and quantitative results are shown.

3.1 Qualitative Results

Fig. 6 shows the results of TM (first row) and FPDTM (second row) methods using the video sequence Dog1. In this video sequence, there is a moving dog plush, sometimes changing its size, nearing and faring to the camera. Note that the TM method loses the dog plush when it changes its size, while the FPDTM is robust to scale changes.

Fig. 5 shows the results of FPDTM (first row) and YOLOV5-StrongSORT (second row) methods using the video sequence Car. In this video sequence, a moving black car changes its perspective through the video. The YOLOV5-StrongSORT detects several objects because this method is a multi-object tracker However, we consider only the black car, marked with an orange bounding box and the label “1 car”. It is observed that the FPDTM loses the object and cannot follow it again when it changes its perspective, while the YOLOV5-StrongSORT tracks it even though it changes its perspective.

Fig. 7 shows the results of TM (first row), FPDTM (second row), and YOLOV5-StrongSORT (third row) methods using the video sequence Person. In this video sequence, a person is still at the beginning of the video, then begins to walk and turn, changing his pose and appearance. The TM only tracks the person in the first video frames while the person is static. FPDTM sometimes loses the person, specifically when this changes its appearance. On the other hand, note that YOLOV5-StrongSORT is more robust in tracking this person.

3.2 Quantitative Results

Three values for the parameter λ: low $\lambda_l = 0.25$, middle $\lambda_m = 0.5$, and high $\lambda_h = 0.75$ were fixed to compute the recall metric. A high value of λ means a better overlapping between the estimated bounding box and the ground-truth bounding box.

There is a class list of objects that the methods based on deep learning can detect. This class list is used for training the deep learning model; thus, these methods can only detect these objects. Within the dataset used in this work, some image sequences have an object that is not within that class list of objects; therefore, the method is incapable of detecting or tracking it.

Table 2 shows the results of the recall metric using the TM, FPDTM, and YOLOV5-StrongSORT methods with different values of λ: $\lambda_l|\lambda_m|\lambda_h$. The highest recall values are shown in bold. The TM method obtains the smallest recalls,

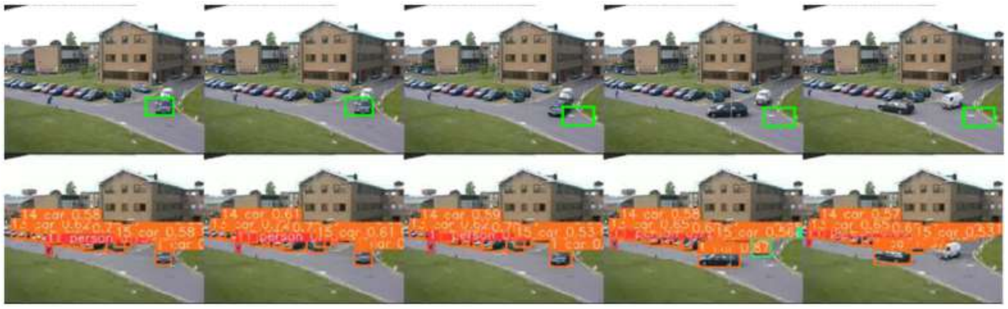

Fig. 5. Results of FPDTM (fisrt row) and YOLOV5-StrongSORT (second row) methods using the video sequence Car

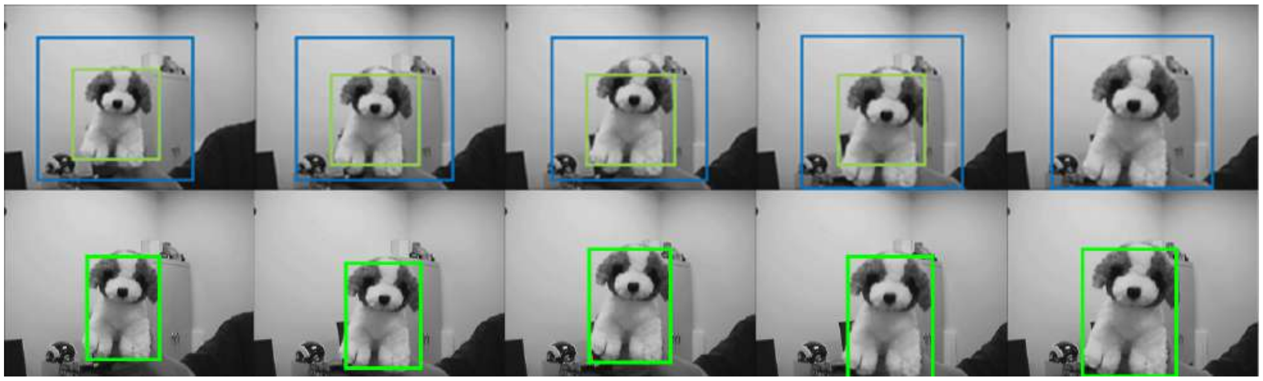

Fig. 6. Results of TM (first row) and FPDTM (second row) methods using the video sequence Dog1

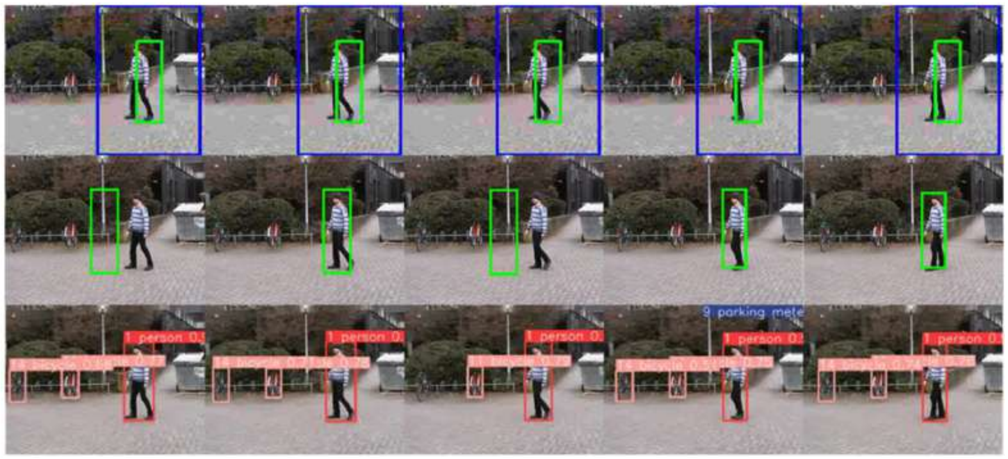

Fig. 7. Results of TM (first row), FPDTM (second row) and YOLOV5-StrongSORT (third row) methods using the video sequence Person

except in the Juice and Jumping videos, where the tracked objects almost do not change their size. The FPDTM method obtains the highest recall values in more videos than the BM and YOLOV5-StrongSORT methods. Note that there are videos such as Board, Box, Juice, Jumping,

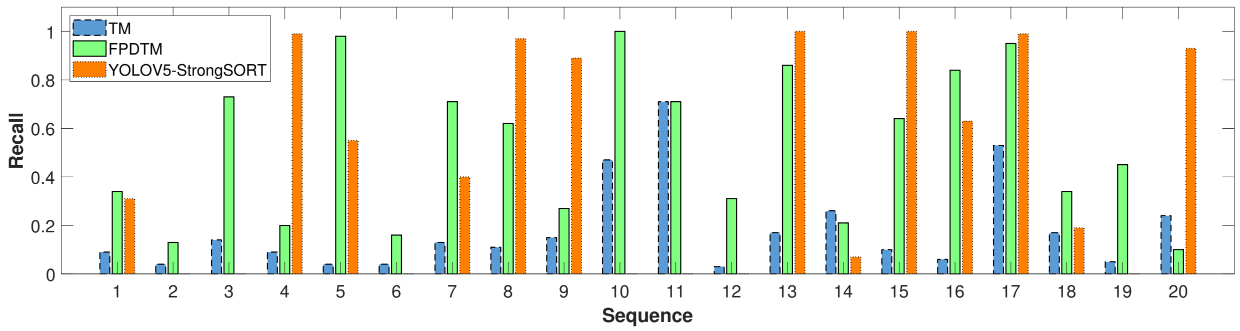

Fig. 8. Results of TM, FPDTM, and YOLOV5-StrongSORT methods

Table 2. Results of the recall metric using the TM, FPDTM, and YOLOV5-StrongSORT methods with different values of λ: $\lambda_l|\lambda_m|\lambda_h$

ID	Sequence	TM	FPDTM	YOLOV5-StrongSORT
1	Ball	$0.09\|0.09\|0.03$	**0.40**|**0.34**|0.19	0.31|0.31|**0.31**
2	Board	$0.05\|0.04\|0.03$	**0.14**|**0.13**|**0.07**	NA|NA|NA
3	Box	$0.14\|0.14\|0.03$	**0.78**|**0.73**|**0.45**	NA|NA|NA
4	Car	$0.18\|0.09\|0.03$	0.46|0.20|0.05	**0.99**|**0.99**|**0.73**
5	Car_2	$0.29\|0.04\|0.03$	**0.98**|**0.98**|**0.93**	0.57|0.55|0.01
6	Carchase	$0.04\|0.04\|0.02$	**0.17**|**0.16**|**0.09**	0.01|0.0|0.0
7	Cup_on_table	$0.13\|0.13\|0.07$	**0.93**|**0.71**|**0.47**	0.40|0.40|0.40
8	Dog1	$0.11\|0.11\|0.09$	0.67|0.62|0.49	**0.98**|**0.97**|**0.51**
9	Gym	$0.18\|0.15\|0.03$	0.60|0.27|0.04	**0.94**|**0.89**|**0.48**
10	Juice	**1.00**|0.47|0.32	**1.00**|**1.00**|**1.00**	NA|NA|NA
11	Jumping	**0.99**|**0.71**|**0.17**	0.81|**0.71**|0.14	NA|NA|NA
12	Lemming	$0.04\|0.03\|0.03$	**0.72**|**0.31**|**0.20**	NA|NA|NA
13	Liquor	$0.17\|0.17\|0.17$	0.86|0.86|0.84	**1.00**|**1.00**|**0.90**
14	Mountain-bike	0.26|**0.26**|**0.12**	**0.49**|0.21|0.11	0.08|0.07|0.00
15	Person	$0.10\|0.10\|0.05$	0.72|0.64|0.35	**1.00**|**1.00**|**1.00**
16	Person_crossing	$0.06\|0.06\|0.06$	**0.86**|**0.84**|**0.75**	0.64|0.63|0.59
17	Person_p_occluded	$0.53\|0.53\|0.53$	**1.00**|0.95|0.82	0.99|**0.99**|**0.99**
18	Singer	$0.17\|0.17\|0.12$	**0.34**|**0.34**|**0.27**	0.23|0.19|0.00
19	Sylvester	$0.05\|0.05\|0.03$	**0.51**|**0.45**|**0.35**	NA|NA|NA
20	Track_running	$0.28\|0.24\|0.11$	0.16|0.10|0.06	**0.93**|**0.93**|**0.57**

Lemming, and Sylvester where the object is not inside the class list of the YOLOV5-StrongSORT training; thus, the YOLOV5-StrongSORT is not applicable (NA). Fig. 8 shows the results of TM, FPDTM, and YOLOV5-StrongSORT methods using λ_m. It is observed that there is a favorable trend for the YOLOV5-StrongSORT because when detecting the object, it does so almost perfectly. All recall values where the YOLOV5-StrongSORT is superior to the other methods are up to 0.8.

On the other hand, in the video sequences where the FPDTM has the highest recall values, the YOLOV5-StrongSORT loses the object, assigning it a new identifier when it is detected again.

4 Conclusion and Future Work

The methods analyzed and implemented in this work offer different results that can be useful for computer vision applications involving object tracking.

The template matching algorithm performs well on objects where lighting changes may occur, and it can follow objects with slight occlusion and smooth movements as long as they do not change their size.

The method based on feature point descriptors and template matching proved good at tracking objects that change size, have scaling, or undergo some slight rotation. However, its weakness was when the object changed shape throughout the video.

The method based on deep learning offered good results when the object belongs to the set of objects previously trained. This method is robust to illumination changes, scaling, partial occlusions, and perspective changes. However, this method can lose the object when this is not detected by severe occlusions, assigning it a different identifier.

The future work proposal is to investigate and implement video object trackers that can have the robustness of a tracker based on deep learning, feature point descriptors, and template matching.

Furthermore, we want to apply it in a real case, such as video surveillance.

The implementation using parallel computing of the method based on feature point descriptors and template matching is also proposed, considering both the multiple cores of a computer and the graphic processing unit.

Acknowledgments

The authors acknowledge the “Investigadoras e Investigadores por México” SECIHTI program and the support from the Supercomputing Center of CIMAT-Mérida “TOOLOK”. The first author acknowledges SECIHTI for funding his postgraduate studies.

References

1. **Ali, A., Jalil, A., Niu, J., Zhao, X., Rathore, S., Ahmed, J., Aksam Iftikhar, M. (2016).** Visual object tracking—classical and contemporary approaches. Frontiers of Computer Science, Vol. 10, pp. 167–188.

2. **ARNES (2013).** VOT Challenge. Website. `https://www.votchallenge.net/`. Accessed 22 Sept 2022.

3. **Broström, M. (2022).** Real-time multi-camera multi-object tracker using yolov5 and StrongSORT with OSNet. Website. `https://github.com/mikel-brostrom/Yolov5_StrongSORT_OSNet`. Accessed 02 Nov 2022.

4. **Broström, Mikel (2023).** BoxMOT: pluggable SOTA tracking modules for object detection, segmentation and pose estimation models. Website. `https://github.com/mikel-brostrom/yolo_tracking`. Accessed 18 Dec 2023.

5. **Corke, P. I., Jachimczyk, W., Pillat, R. (2011).** Robotics, vision and control: fundamental algorithms in MATLAB, Vol. 73. Springer.

6. **Du, Y., Zhao, Z., Song, Y., Zhao, Y., Su, F., Gong, T., Meng, H. (2023).** StrongSORT: Make DeepSORT great again. IEEE Transactions on Multimedia.

7. **Elgendy, M. (2020).** Deep learning for vision systems. Manning Publications, Shelter Island, NY.

8. **Lee, D.-H. (2021).** CNN-based single object detection and tracking in videos and its application to drone detection. Multimedia Tools and Applications, Vol. 80, No. 26-27, pp. 34237–34248.

9. **Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C. L. (2014).** Microsoft coco: Common objects in context. Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13, Springer, pp. 740–755.

10. **Maggio, E., Cavallaro, A. (2011).** Video tracking: theory and practice. John Wiley & Sons.

11. **Mangawati, A., Leesan, M., Aradhya, H. R., et al. (2018).** Object tracking algorithms for video surveillance applications. 2018 international conference on communication and signal processing (ICCSP), IEEE, pp. 0667–0671.

12. **Padilla, R., Netto, S. L., Da Silva, E. A. (2020).** A survey on performance metrics for object-detection algorithms. 2020 international conference on systems, signals and image processing (IWSSIP), IEEE, pp. 237–242.

13. **Pat-Chan, A. E., Hernandez-Lopez, F. J., Moreno-Sabido, M. R. (2023).** Seguimiento de objetos en video mediante el método de emparejamiento de bloques.. Res. Comput. Sci., Vol. 152, No. 8, pp. 35–46.

14. **Pernici, F., Del Bimbo, A. (2013).** Object tracking by oversampling local features. IEEE transactions on pattern analysis and machine intelligence, Vol. 36, No. 12, pp. 2538–2551.

15. **Trucco, E., Plakas, K. (2006).** Video tracking: a concise survey. IEEE Journal of oceanic engineering, Vol. 31, No. 2, pp. 520–529.

16. **Ultralytics (2021).** YOLOv5: A state-of-the-art real-time object detection system. Website. `https://docs.ultralytics.com`. Accessed: 21 Aug 2023.

17. **Wojke, N., Bewley, A., Paulus, D. (2017).** Simple online and realtime tracking with a deep association metric. 2017 IEEE international conference on image processing (ICIP), IEEE, pp. 3645–3649.

18. **Zhou, J., Yao, Y., Yang, R. (2022).** Deep learning for single-object tracking: A survey. 2022 IEEE 2nd International Conference on Software Engineering and Artificial Intelligence (SEAI), IEEE, pp. 12–19.

Article received on 15/01/2024; accepted on 13/01/2025.
**Corresponding author is Andrés E. Pat-Chan.*