International Journal of Intelligent Robotics and Applications (2020) 4:191–201
https://doi.org/10.1007/s41315-020-00137-0

REGULAR PAPER

Auto-correct-integrated trackers with and without memory of first frames

Ali Sekhavati[1] · **Najmeh Eghbal**[1]

Received: 3 December 2019 / Accepted: 15 May 2020 / Published online: 2 June 2020

Abstract
Visual short-term tracking in a long sequence of images with a lot of pose variations, target deformations and different types of occlusion is one of the harshest tasks in image processing. Overcoming such challenges can be performed in a superior way by combining different trackers. In this paper, we propose a method that combines different algorithms for tracking the given target. The algorithms are KCF (Henriques et al. in IEEE Trans Pattern Anal Mach Intell 37(3):583–596, 2014), MIL (Babenko et al. in Visual tracking with online multiple instance learning. In: 2009 IEEE conference on computer vision and pattern recognition. IEEE, 2009), CSR-DCF (Lukezic et al. in Discriminative correlation filter with channel and spatial reliability. In: Proceedings of the IEEE conference on computer vision and pattern recognition, 2017) and MOSSE (Bolme et al. in Visual object tracking using adaptive correlation filters. In: 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. IEEE, 2010) trackers. In this method, these algorithms can continuously correct each other's faults. They also implicitly search for the target even when they are tracking it, to make sure the object they are tracking is the real target. Thus, they outperform many of the state-of-the-art trackers as well as their components when they work independently. In this paper, we examine three trackers which we made in such way. Two of these trackers run at speeds close to real-time on a CPU and so we compare them with some famous wide-spread tracking algorithms both in terms of accuracy and speed.

Keywords Visual object tracking · Correlation filters · Combined trackers · Self-correcting tracking

1 Introduction

The controversial task of short-term visual object tracking in any sequence of images has always been an important one through the history of image processing with so many different applications in robotics, artificial intelligent, traffic control, shop and factory security etc. Background clutters, partial and full occlusion, illumination variation, deformation, camera motion and many other variations can cause serious problems for trackers' performance (Wu et al. 2015). Most of the recent tracking algorithms are based on deep learning approaches. Even though they can address these problems, they are computationally expensive.

One of the oldest and yet most famous ways of tracking an object in any sequence of images is tracking via Kalman filter (Kalman 1960). In control system theory, Kalman filter can be used for linear systems to estimate their states when noise has a Gaussian distribution and the filter is based on prediction and correction parts (Kalman 1960; Patel and Thakore 2013; Yilmaz et al. 2006). In image processing, it can be used for object tracking and predicting the next position of the target. Another method for object tracking is through the Mean Shift algorithm, which shifts every data point to the mean of data points in its neighborhood iteratively (Cheng 1995). There is also the CAMSHIFT (Continuously Adaptive Mean Shift) algorithm that develops the Mean Shift algorithm so that it can deal with probability distributions which change and move dynamically in time, and make it able to track moving objects in video frames (Bradski 1998a, b). In this method, one can set the color of the target to be tracked.

For having a more robust tracker, a lot of researchers tried to extract the features of the target during the process and learn from them. In (Grabner et al. 2006), by using classifiers, an algorithm is designed to track the target in real-time as well as on-line learning in order to handle appearance changes and separate it from the background. A long term

✉ Ali Sekhavati
al.sekhavati903@sadjad.ac.ir

[1] Sadjad University of Technology, Mashhad, Iran

tracker is represented by Kalal et al. (2011) which is consisted of three parts: tracking, learning and detection. The tracker follows the target in every frame whereas the two other parts try to minimize its failures.

Many of the new tracking algorithms are based on deep neural networks and convolutional ones. Bertinetto et al. (2016) introduced a tracking algorithm with a novel fully-convolutional Siamese network, which despite its great simplicity and being made based on deep learning algorithms, it outperforms many short-term and real-time trackers both in speed and accuracy. He et al. (2018) designed a twofold Siamese network for real-time object tracking. Their network benefits from the heterogeneity of semantic features and appearance features and by designing a channel attention module to achieve target adaptation, they achieved great results both on OTB and VOT benchmarks. Lee et al. (2018) proposed a long term tracker using deep convolutional features and a human-based memory model for storing the features of the target. They also used a detector to realize the existence of the object and find it when it is not followed by the tracker. They divided the memory management part into a short term memory and a long term memory, each with limited capacity for storing the target's specific Siamese and semantic features.

Although most of the trackers which use deep learning algorithms are more accurate and robust than the rest, they usually have been unable to track their target in real-time applications. Held et al. (2016) created one of the earliest real-time trackers using neural networks. The method they used was not based on on-line learning but off-line training instead. An example of fast short-term object tracker with high accuracy and robustness based on Siamese network was performed by Li et al. (2018), achieving the processing speed of 160 frames per second and leading performance in VOT2015 (Kristan et al. 2015), VOT2016 (Kristan et al. 2016) and VOT2017 (Kristan et al. 2017) in the real-time challenges.

Some trackers use adaptive, kernelized or discriminative correlation filters such as Henriques et al. (2014), Bolme (2010) and Lukezic et al. (2017). Since the method used in this article is based on the last three works, in the next part they will be discussed with more details. Bochinski and Eiselein (2017) created an algorithm for tracking objects without using image information by considering the overlap of detections between time steps. By using this simple and efficient method, they achieved the tracking speed of 100 K fps and their algorithm outperformed state-of-the-art trackers on the DETRAC vehicle tracking dataset.

Some other tracking algorithms are combination of other methods. Akshay et al. (2016) made a tracker combining Kalman filter, Gaussian mixture model and Optical flow in order to achieve better tracking result. Tran (2015) used a combination of Generalized Hough Transform, and Particle filter in order to make a tracker robust to appearance variations such as scaling, rotation, non-rigid deformation or illumination changes.

In this paper, we propose a method for combining different trackers and then we introduce an algorithm in which they can minimize their faults and correct each other. We also implicitly search for the target when we are tracking it to make sure the real target is being tracked rather than any other similar object. In this way, we can track any object through literally challenging obstacles such as pose variation, target deformation and different types of occlusion with or without similar objects around for short-term visual tracking.

The idea of this method arose noticing the fact that if many trackers follow one object at the same time, as long as they are doing their job correctly, their regions of interest must be close to each other and have similar coordinates. But when one of those trackers offers a different bounding box than that of many other trackers, it probably has failed recognizing the target and must be corrected. With this intuitive and reasonably important assumption in mind, we propose two trackers ACI (Auto-Correct-Integrated) and ACI with memory of first frames.

In this new method, we use three tracking algorithms to estimate the position of the target in every picture. Each of these three is good in its own way and fails toward different reasons. The trackers used in the ACI algorithm are KCF (Henriques et al. 2014), MOSSE (Bolme 2010) and CSR-DCF (Lukezic et al. 2017) which is also known as CSRT. All these trackers are based upon correlation filters which can have a much faster performance than many other methods. More information on how they track, their main qualities and their practical issues for each of them is available in the next part.

Many other researchers have tried to combine different tracking algorithms such as Akshay et al. (2016) and Tran (2015). We tested our method on a database containing 7355 frames with different targets to track in different areas and then compared the performance of these three new trackers with some others given the same bounding boxes in the first frame. All of our trackers outperform their components and also all the trackers in OpenCV in terms of accuracy while being computationally more efficient and less complex than many of them.

In the following, we try to clearly describe our method with enough details in Sect. 2. In Sect. 3 after that we evaluate our trackers with their components and also eight OpenCV trackers. The results of this comparison is collected in Table 1. Finally, in conclusion we briefly overview the advantages of this method.

2 Proposed method

In this work we use four tracking algorithms and design a method so that these trackers can evaluate each other's performance and correct each other for overcoming a bigger range of challenges. Three of these methods use correlation filters and one of them uses multiple instant learning for training a classifier to separate the target from background. Each of them have specific strengths and weaknesses and in this new combination of them, we tried to use their advantages to cover their weaknesses.

A. MOSSE tracker:

While being the fastest one among all these four trackers, it is able to handle variations in lighting, scale, pose, and non-rigid deformations. The tracker can follow the object given to it at speeds over 1.4 K FPS using an Intel Core i5-8250U CPU and 8 Gigabytes of RAM. According to our experiments, its robustness is not as good as the CSRT, but yet better than KCF tracker. This one also has a good performance when it comes to redetecting the target after full occlusions even sometimes in the presence of similar objects around.

The tracker is based upon adaptive correlation filters for modeling the target, and convolutional filters in order to track. It operates on the gray-scale images and if the images given are not so, it converts them to that form. The tracker uses the model to find the target in the new frame and then it updates the filter. For speeding up the process, the algorithm computes the correlation in the Fourier domain as Fast Fourier Transform (FFT).

B. KCF tracker:

In terms of accuracy and drawing a bounding box which contains the whole target without assigning too many other pixels of the image as the target and fitting it the most, the KCF tracker performs better than the other three. Using the same system mentioned above for the MOSSE tracker, its tracking speed is around 150 fps. Although it may not be as robust as the other ones, its high speed, accuracy and the ability to handle full occlusions can come handy for correcting the other trackers in a short time.

According to (Henriques et al. 2014), for tracking the target, the tracker trains a model with an image patch which is larger than the target which provides more context, by having the bounding box given in the first frame. Then, after getting the next frame, they look for the pixels having the most similarity to the previous patch. After that, the algorithm trains a new model having the information of both the new position and the previous one. In this way, the data of all images are used.

C. CSR-DCF tracker

This is also known as CSRT among trackers of OpenCV (Open Source Computer Vision Library). According to our tests, this one can handle partial occlusion, change in object appearance, scale and posing variations. But it is much slower than KCF and MOSSE with processing speed of almost 29 fps using the same system as the other three trackers. Despite its unhurried performance, it was the most robust one not only in comparison with the other two, but also among the rest of OpenCV trackers. The problems we faced working with this tracker other than its speed were for the full occlusions having similar objects close to the target. Also it sometimes drew inaccurate bounding boxes when the number of frames got too many and there was also a lot of pose variation.

This tracker is based upon two main steps: the localization step and the update step. In the first step, using the correlation filters by having the position of the object in the previous frame and image patch features, the location of the target in the new frame is estimated. After that, it estimates the detection reliability by using Per-channel filter responses. Finally, scale is being estimated by a correlation filter as in Danelljan et al. (2014). In the next step, which is the updating step, we extract and update the foreground and background histograms, estimate the reliability map and use it to estimate a new filter and update it. After that, we estimate the learning channel reliability, and then calculate channel reliability and update it. More information can be acquired through (Lukezic et al. 2017).

D. MIL tracker:

This tracker can handle partial occlusions better than KCF and MOSSE, but does not have a good performance handling full occlusions or redetecting targets. It is slower than CSRT but it is more robust than KCF and MOSSE. Similar to KCF and MOSSE, it does not change the scale of given bounding box. It loses the target when it is occluded either by a similar object or when there is a similar object close to the target.

This tracker takes the given bounding box and computes the feature vectors. Then it uses a classifier to estimate the target. After updating the target's location, it updates its appearance model (classifier) by labeling a bag of the target's features positive and the area around them negative. By extracting a bag of positive examples instead of only one positive example in each frame, this method can better train its classifier to address occlusions. That is how Multiple Instance Learning avoids the drift problem for a robust tracking (Babenko and Yang 2009).

E. Our method:

The basis of our algorithm is by taking advantage of the fact that in the first frame, we give the same bounding box to all these three trackers. Each of these trackers extract different features from the target and background. Thus, they offer different bounding boxes for the location of what they follow in the next frame. Having the exact coordinates of all bounding boxes, we define an average bounding box and consider it as the main bounding box. After that, we measure the distance between the bounding boxes proposed by the trackers and the average bounding box. Now if each of these trackers propose a bounding box for an object far from the average bounding box or finds its tracking reliability lower than its threshold, we conclude that this tracker's results cannot be trusted and it needs correction. The main reason for the success of this method is that the algorithm continuously corrects its trackers. So if each one of them fails, ACI corrects it and it can later correct ACI by correcting its other trackers too.

For correcting them, first we calculate the average box without considering the wrong tracker, which means we only calculate the mean of the other boxes. After that, if a specific number of frames have passed and the tracker has not redetected the target yet, we reinitialize the wrong one and give it the coordinates of the new average box. That tracker will treat the new bounding box as a new object which must be tracked. But we also keep the information extracted in the first frames until a specific number of mistakes of all trackers in order to have more features to look for. In this way, despite much bigger changes in the appearance or pose, the target can still be tracked.

Sometimes three of the trackers suddenly make different mistakes and either get far different results from the main box, or have tracking reliabilities lower than their thresholds. In such cases, if the problem is caused by having tracking reliabilities lower than threshold, since each of those wrong trackers are trying to redetect the target on their own, we give them a limited number of frames to try to find it in them. While they are looking for the target, the average bounding box is considered the bounding box of the only tracker which has tracking reliability higher than its threshold. If even after that number of frames they all fail, they get reinitialized and get the object in the main box as the new object to track. The reason that we did not quickly try to fix them all is that such cases usually happen due to full occlusion, and that one tracker which has tracking reliability more than its threshold is wrong. So we give them some time to keep searching for the target in the coming frames and when we are almost able to conclude this situation has not occurred because of full occlusion, we reinitialize those trackers. We do not choose the searching time to be too long too. Since if the situation has not happened out of full occlusion, then it means that tracking the target has become really difficult. In such cases, if we do not reinitialize them quickly, that one tracker which is successfully doing its job might lose it as well.

Also there are times when each of them are tracking different objects with different coordinates and all are having tracking reliabilities higher than their thresholds without even two of them offering close bounding boxes. For such situations we set the main box to be the bounding box of the CSR-DCF tracker but we do not reinitialize the other ones. Because according to our experiments, even though CSR-DCF is the most robust one it is not immune to mistakes and it fails handling full occlusions which others do not and those they are able to correct it. In such cases, the wrong ones will slowly find their tracking reliabilities below the threshold and they will start searching for the target. That is just like the condition mentioned above which was the time when only one tracker has its tracking reliability above its threshold. A briefer and clearer description of the process is summarized in Algorithm 1 and Fig. 1.

Algorithm 1: The Auto-Correct-Integrated (ACI) tracker with memory of first frames algorithm using CSR-DCF, KCF and MOSSE

```
Initial settings:
        TR = Threshold of trackers tracking reliability
        Counter = 0
        C = Height, width and distance difference from the main bounding box
        T = The set threshold for C
        NF = Number of frames which trackers can track alone before calling the other ones
        TTE = Total number of trackers' errors
        TTET = Total number of trackers' errors threshold

Inputs:
        New frame
        The previous location of the target (previous main box)
        The trackers previous bounding boxes
        Trackers tracking reliability in the last frame (TRL)
Outputs:
        New main box
        New trackers bounding boxes if available

 while (the frame is not the last frame)
    {Get the new frame
    Update all trackers by having their previous bounding boxes
    if (C < T in all trackers)
        {main box = mean of all the bounding boxes
        Counter = 0}
    else if (C < T in two trackers)
        {main box = mean of those two trackers' bounding boxes
        Reinitialize the wrong one and give it the main box to track
        Counter = 0}
   else if (TRL > TR in only one tracker)
        {main box = bounding box of that tracker
        Counter = Counter + 1
        Other trackers search for the features extracted in the first frames and their last track for redetecting the
        target
        //In ACI without memory of first frames, trackers only look for the extracted features in their last track
        if (Counter >= NF)
            {Reinitialize other trackers giving them the main box to track}}
   else if (TRL > TR and C >T in all trackers)
            {main box = bounding box of CSR-DCF
            Counter = 0}
     if (TTE < TTET)                    //This if statement is not implemented in ACI without memory of first frames
        {Store extracted features}  //This if statement is not implemented in ACI without memory of first frames
     Draw the main box}
```

In the algorithm above, we update the trackers in each frame and calculate the average bounding box. If the height, width and coordinates of bounding boxes for each tracker is close enough to the average bounding box, the main box is the average bounding box. C, T, TTE and TTET are calculated in the following manner:

$$C = \left\{ \left|X_{tracker} - X_{average}\right|, \left|Y_{tracker} - Y_{average}\right|, \left|W_{tracker} - W_{average}\right|, \left|H_{tracker} - H_{average}\right| \right\} \quad (1)$$

$$T = \left\{ \alpha \times \left(\frac{H_{average} + W_{average}}{2} \right), \alpha \times \left(\frac{H_{average} + W_{average}}{2} \right), \alpha \times W_{average}, \alpha \times H_{average} \right\} \quad (2)$$

Fig. 1 ACI with memory of first frames in practice. Black box: Main box—Blue box: KCF's Box—Red box: MOOSE's Box—Green box: CSR-DCF's box. ACI keeps tracking the red runner until the yellow runner occludes the target and it mistakenly tracks the yellow one but after a few frames, it redetects the red runner by using its memory and continuous tracking him (color figure online)

$$TTE = \sum_{i=1}^{n} \sum_{j=1}^{4} \begin{cases} 1, & C_{i,j} > T_{i,j} \\ 0, & C_{i,j} < T_{i,j} \end{cases} \tag{3}$$

$$\text{TTET} = \beta \times \text{n}. \tag{4}$$

where X and Y are bounding boxes' coordinates, H and W are its height and width, α is distance rate, n is number of trackers and β is a constant. $C_{1,1}$ and $C_{1,2}$ correspond to the distance of the first tracker's bounding box from main box. $T_{1,1}$ and $T_{1,2}$ are distance thresholds of the first tracker's box from the main box. $C_{1,3}$ and $C_{1,4}$ demonstrate the size difference between the first tracker's bounding box and the main box. $T_{1,3}$ and $T_{1,4}$ state the thresholds for difference in first tracker's bounding box size than the main box. For each one of the trackers we count the number of mistakes and then we calculate the total number of mistakes for all trackers thorough Eq. (3). TR and TRL are calculated in different ways for each of these trackers and it is better not to change their default way of calculating them. When α is set too high, it decreases the ACI algorithm's ability to detect and correct the wrong trackers. When it is set too low, ACI keeps reinitializing the trackers that are relatively correct, which leads to having less tracking features.

Then if two of the trackers' bounding boxes are close to the average bounding box, we set the main box as the average of these two trackers and correct the false one. If only one tracker has its tracking reliability over its set threshold, then the main box is that tracker's bounding box. Then we wait for other trackers to search and find the target in the next NF frames; which means we both search and track. If after NF frames they fail to find the target, we reinitialize them and give them the main box to track. NF is chosen based on experience. If trackers have bounding boxes too far from each other, we set the CSR-DCF's bounding box as the main box. This is mainly because in this case we must choose one tracker over the other ones, and since in this work we valued robustness over redetection, we chose CSR-DCF's bounding box. One can choose other trackers' bounding boxes to increase other capacities of the algorithm. But we should keep in mind that because this condition happens only when all trackers are tracking different objects, it occurs fewer times than other else if blocks. Finally, if we are in the beginning of tracking and TTE is not higher than TTET, we store features extracted from the target so that later on trackers can look for extracted features from both the first and last frames. It must be noticed that the reason why we do not store the features of the last tracking frames is that the trackers do it implicitly during their updates.

Next, we examine how we can combine four trackers by ACI algorithm. As the number of trackers increases, the algorithm gets more complicated due to the frustrating increase in the number of if-statements. For simplicity and better performance, it is advised to combine trackers in the following manner:

First we locate the area with highest number of bounding boxes close to each other. Then in each frame for those trackers that have tracking reliability below their set threshold or they have a bounding box too far from the main box we only increase their mistakes number by one and calculate the main box without considering them. It is important not to reinitialize them quickly because whenever we reinitialize each of them, that tracker loses its data. Only when their mistakes numbers have achieved their thresholds, we can reinitialize the trackers.

Algorithm 2: The Auto-Correct-Integrated (ACI) tracker without memory of first frames algorithm using MIL, CSR-DCF, KCF and MOSSE

```
Initial settings:
        TR = Threshold of trackers tracking reliability
        C = Height, width and distance difference from the main bounding box
        T = The set threshold for C
        TTM = Total number of a tracker's errors
        β = Total number of a tracker's errors threshold
Inputs:
        New frame
        The previous location of the target (previous main box)
        The trackers previous bounding boxes
        Trackers tracking reliability in the last frame (TRL)
Outputs:
        New main box
        New trackers bounding boxes if available

while (the frame is not the last frame)
    {Get the new frame
    Update all trackers by having their previous bounding boxes
    if (C < T in all trackers)
        {main box = mean of all the bounding boxes
        TTE of all trackers = 0}
    else if (C < T in three trackers)
        {main box = mean of those three trackers' bounding boxes
        TTM of that wrong tracker = TTM of that wrong tracker + 1
        If (TTM of that wrong tracker >= β of that wrong tracker)
        {Reinitialize that wrong tracker
        TTM of that wrong tracker = 0}}
    else if (C < T in two trackers)
        {main box = mean of those two trackers' bounding boxes close to the main box
        TTM of those two wrong trackers = TTM of those two wrong trackers + 1
        if (TTM of any of the wrong trackers >= β of that tracker)
        {Reinitialize that wrong tracker
        TTM of that wrong tracker = 0}}
    else if (TRL > TR in only one tracker)
        {main box = bounding box of that tracker
        TTM of those three wrong trackers = TTM of those three wrong trackers + 1

        if (TTM of any of the wrong trackers >= β of that tracker)
        {Reinitialize that wrong tracker
        TTM of that wrong tracker = 0}}
    else if (TRL > TR and C > T in all trackers)
        {main box = bounding box of CSR-DCF}
    Draw the main box}
```

In this algorithm, for each tracker TTM is calculated as the following:

$$\mathrm{TTM} = \sum_{j=1}^{4} \begin{cases} 1, & C_j > T_j \\ 0, & C_j < T_j \end{cases}. \tag{5}$$

More trackers can be added using the algorithm 2 just by adding the number of else-if blocks.

Since we use only the output of those tracking algorithms, any tracker can be replaced with the ones used in ours. This can clearly result in different performances and by implementing this method, if we choose trackers in a way which can extract a bigger variety of features, then we can have a more robust and accurate tracker. The code is written in pure Python and is publically available at (https://github.com/

AliSekhavati/ACI-trackers/blob/master/README.md) as well as the videos.

3 Evaluation

This part focuses on practical issues of the ACI tracker and how and why it has become different than its components. We first compare the performance of the ACI tracker with the trackers used to create it just to show how better the new one can track and why it does not make some of the mistakes which they might make while they are tracking targets alone. After that, we compare it with some of the OpenCV trackers both in speed and accuracy to have a better understanding of other trackers as well.

A. Comparison with MOSSE tracker

MOSSE is the fastest of all these four due to using Fast Fourier Transform, correlation filters and the tracker's simple architecture. But according to our experiments, when it comes to dealing with big occlusions which might cover around one-third of the target, its tracking reliability decreases below its threshold until that partial occlusion is passed. That is for the big change in the target which does not fit the tracker's filter. Then it may redetect the target. If this type of occlusion happens while pose variation is occurring, such as what happens in "nascar_02" video, it totally stops tracking until it redetects the same or a similar pose of the target, which might not happen in many cases. Because the filter was made based on the another pose of the target which cannot be seen after pose variation and occlusion. Our method successfully continued tracking every time by using CSR-DCF and MIL's robustness.

Another problem that MOSSE tracker suffers from is its dependency on the target's high level of appearance change in the first frames. In other words, the appearance of the target must not change suddenly in the beginning frames which the algorithm has just started learning. The reason for this is because this tracker is mainly depended on correlation filters which cannot handle big appearance changes at first. Such problem was observed tracking five out of eight runners in the "race" video and it caused the tracker to fail at the very beginning and never be able to find the target again. While in the ACI tracker, given the exact same bounding box at the first frame, it successfully tracked all the runners in that video. The CSR-DCF and MIL's robustness and the accuracy of KCF combined with the redetecting ability of KCF and MOSSE were big helps dealing with these type of issues.

This tracker is not immune to big pose variation during scale change of the target for the same reasons mentioned above. In "nascar_01" it tracks the target for 81 frames out of 135 and then fails according to this problem. Also, not being able to make big changes in the size of bounding box, for almost 20 of those 81 frames it only tracks half of the car.

B. Comparison with KCF tracker:

KCF has difficulty dealing with partial occlusion. For instance, in "nascar_02" video, when the followed car passes a thin column, the KCF suddenly loses it and fails in redetection phase. This problem is caused by its too high redetection threshold. Whereas in ACI tracker, it tracks the car during different partial occlusions in the whole movie because of CSR-DCF and MIL's performance during such occlusions and MOSSE's performance afterward.

Such occlusions affect the tracker's performance in "american_pharoah" and "race" videos too. In the former, an occlusion occurs after 35 frames and KCF which was properly tracking the target, stops tracking and fails in redetection phase for the rest of the video. But in ACI, because of the MOSSE's better redetection ability and CSR-DCF's continuous tracking ability, KCF quickly gets corrected by the other three algorithms and keeps accurately tracking it in the whole video.

In the "race" video, showing a better performance than MOSSE in some cases, it handles the redetecting difficulties for most of the time other than the final frames and thus, tracks the target in 79 frames out of 339. The occlusions were mostly for the runners who were covering some parts of the target runner. While it does prove a better performance than many other new trackers, the ACI tracker outperforms this by accurate tracking in more than 325 frames.

In "nascar_01" video, it follows the target in the first 67 frames but after that, the camera zooms in while the target car is passing the camera and so pose variation takes place as well, which makes it pretty difficult to track using correlation filters alone. The ACI tracker does track it in all the 136 frames. But since none of its component trackers are adaptive enough to the scale variation, it fails in drawing a bounding box covering the whole car, and just approximately considers half of the car instead.

C. Comparison with CSR-DCF tracker

Sometimes there are objects similar to the target when the it gets occluded. This occlusion can take place by that resembling object too. In such cases, CSR-DCF tracker can easily get confused and misses the real target following the similar one while KCF and MOSSE tracking methods can come in hand and correct it if possible. For instance, in the "american_pharoah" video, in some experiments we set the bounding box to be only the body of horses. During occlusions, the CSR-DCF

tracker lost the target and started following the body of the horse behind or other horses until the end of the video. That is because its tracking reliability did not get below its threshold and so it tracked the wrong object. But this issue also causes more robustness for the tracker when it is facing other problems. In ACI tracker, after such occlusions the algorithm restarts tracking the real target thanks to the KCF and MOSSE redetection phase. For this video, when bounding box is just the body of the horse, the CSR-DCF tracker fails because of occlusions and other ones quickly correct it. But in that video, when the bounding box is the whole horse and its rider, CSR-DCF tracks it and the only problem was proposing a bounding box bigger than target. This was caused by long sequence of frames.

In the "race" video tracking one of the runners, after huge partial occlusion, CSR-DCF mistakenly tracks the runner who was in the way of the target. But MOSSE and KCF redetect the real target, fix CSR-DCF and continue tracking it.

D. Comparison with MIL tracker

MIL tracker can handle big partial occlusions better than the other three. But it usually fails handling full occlusions. The main reason for this problem is that during full occlusions, the tracker still tries to update its classifier by new online features it extracts. However, because during full occlusions the target is not visible the classifier of the tracker gets updated with wrong online features. Thus, after full occlusions there is a high probability that the tracker tracks the occluding object instead of the target. In "american_pharoah" video it first tracks the target during huge partial occlusions that none of the other three trackers do it. This success is for the trackers multiple online learning. In the "race" video anytime another runner fully or partially occludes the tracking runner, this tracker's classifier mistakenly considers the occluding runner instead of the real target. Figure 2 shows how ACI algorithm with 4 trackers works in practice.

E. Statistically comparison with other trackers

In this part, we generally compare the ACI tracker in speed and accuracy with the trackers implemented in OpenCV for being able to briefly compare all these trackers at once. For this purpose, we examine the ability of Boosting, MIL, KCF, TLD, Median Flow, GOTURN, CSRT (CSR-DCF), MOSSE and our proposed method (ACI) as Table 1 illustrates.

As the table above demonstrates, the ACI trackers both using and not using the data of the first frames outperform all the trackers implemented in OpenCV in terms of accuracy and two of them are faster than many others. We used python version of these trackers and gathered the information above by tracking different targets in 7355 frames in 4 videos. This table shows adding the fourth tracker which is fundamentally different increases the algorithm's capacity for better addressing different problems, resulting an accuracy much better than adding first frames' memory. The only problem is the decrease of the average speed which is caused by MIL's low processing speed. Another thing which can be understood from results of KCF tracker is that it has a really high accuracy when it detects the target, but most of the time it suffers from a poor performance in the detection phase. For overcoming the issue, this tracker's reliability threshold must decrease. We also faced a problem using the GOTURN tracker and it was for the

Fig. 2 ACI algorithm with 4 trackers, correcting its components and its components correcting it. Black box: Main box—Blue box: KCF's Box—Red box: MOOSE's Box—Green box: CSR-DCF's box—white box: MIL's Box (color figure online)

Table 1 Comparing ACI trackers with eight OpenCV trackers

Tracker's name	Average processing speed (FPS)	Frames with tracking reliability over threshold (%)	Accuracy when target is detected (%)	Overall accuracy (%)
Boosting	26.32	100	6.42	6.42
MIL	13.55	100	44.19	44.19
KCF	143.13	11.63	98.87	11.17
TLD	7.97	97.43	11.58	11.36
Median flow	109.98	72.54	12.86	7.07
GOTURN	11.07	100	33.33	33.33
CSR-DCF	29.27	99.43	56.70	56.68
MOSSE	1450.46	82.71	94.92	78.52
ACI without memory of first frames (3 trackers)	22.59	100	87.19	87.19
ACI with memory of first frames (3 trackers)	20.29	100	88.88	88.88
ACI without memory of first frames (4 trackers)	8.49	100	93.10	93.10

Fig. 3 Comparison with some other trackers: ACI algorithm with 3 trackers without memory of first frames. Black box—Boosting tracker: Red box—MIL tracker: Purple box—KCF tracker: Cyan box—TLD tracker: Orange box (color figure online)

times when it was unable to separate the object from background. In such cases, its region of interest would keep expanding and it made the task pretty challenging. For instance, in the "american_pharoah" video that is consisted of 3183 frames, the processor could only compute in the first 211 frames and after that it stopped the program for being computationally too expensive. In the first frames, the ACI with memory is saving data and so its processing speed is almost as half as ACI. But after a while when it has saved the features required, it achieves the speed of normal ACI. Figure 3 compares Boosting, MIL, KCF, TLD and ACI algorithm with 3 trackers without memory of first frames.

4 Conclusion

In this article we proposed a method of combining trackers for using strengths of them all covering each other's weaknesses. In this method trackers can correct each other and thus, they are able to deal with more difficult obstacles. The results of such combination both using and not using the features extracted in the first frames outperformed all the trackers used for making the ACI and also all the other trackers in OpenCV. Since any tracker can be combined in this way, it seems to be a good idea to use different feature extractors together such as algorithms based on neural networks or other methods mentioned in Sect. 1.

References

Akshay, S., Sajin, T., Ram Prashanth, A.: Improved multiple object detection and tracking using KF-OF method. Int. J. Eng. Technol. **8** (2016)

Babenko, B., Yang, M.-H., Belongie, S.: Visual tracking with online multiple instance learning. In: 2009 IEEE Conference on Computer Vision and Pattern Recognition. IEEE (2009)

Bertinetto, L., et al.: Fully-convolutional siamese networks for object tracking. In: European Conference on Computer Vision. Springer, Cham (2016)

Bochinski, E., Eiselein, V., Sikora, T.: High-speed tracking-by-detection without using image information. In: 2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS). IEEE (2017)

Bolme, D.S., et al.: Visual object tracking using adaptive correlation filters. In: 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. IEEE (2010)

Bradski, G.R.: Computer vision face tracking as a component of a perceptual user interface. In: Workshop on Applications of Computer Vision, Princeton, NJ (1998–10) (1998a)

Bradski, G.R.: Real time face and object tracking as a component of a perceptual user interface. In: Proceedings Fourth IEEE Workshop on Applications of Computer Vision. WACV'98 (Cat. No. 98EX201). IEEE (1998b)

Cheng, Y.: Mean shift, mode seeking, and clustering. IEEE Trans. Pattern Anal. Mach. Intell. **17**(8), 790–799 (1995)

Danelljan, M., et al.: Accurate scale estimation for robust visual tracking. In: British Machine Vision Conference, Nottingham, September 1–5, 2014. BMVA Press, (2014)

Grabner, H., Grabner, M., Bischof, H.: Real-time tracking via on-line boosting. Bmvc **1**(5), 6 (2006)

He, A., et al.: A twofold siamese network for real-time object tracking. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (2018)

Held, D., Thrun, S., Savarese, S.: Learning to track at 100 fps with deep regression networks. In: European Conference on Computer Vision. Springer, Cham (2016)

Henriques, J.F., et al.: High-speed tracking with kernelized correlation filters. IEEE Trans. Pattern Anal. Mach. Intell. **37**(3), 583–596 (2014)

Kalal, Z., Mikolajczyk, K., Matas, J.: Tracking-learning-detection. IEEE Trans. Pattern Anal. Mach. Intell. **34**(7), 1409–1422 (2011)

Kalman, R.E.: A new approach to linear filtering and prediction problems. J. Basic Eng. **82**(1), 35–45 (1960)

Kristan, M., et al.: The visual object tracking vot2015 challenge results. In: Proceedings of the IEEE international conference on computer vision workshops (2015)

Kristan, M., et al.: The visual object tracking vot2016 challenge results. In: ECCV Workshop, vol. 2. no. 6. (2016)

Kristan, M., et al.: The visual object tracking vot2017 challenge results. In: Proceedings of the IEEE International Conference on Computer Vision (2017)

Lee, H., Choi, S., Kim, C.: A memory model based on the siamese network for long-term tracking. In: Proceedings of the European Conference on Computer Vision (ECCV). (2018)

Li, B., et al.: High performance visual tracking with siamese region proposal network. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (2018)

Lukezic, A., et al.: Discriminative correlation filter with channel and spatial reliability. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. (2017)

Patel, H.A., Thakore, D.G.: Moving object tracking using kalman filter. Int. J. Comput. Sci. Mob. Comput. **2**(4), 326–332 (2013)

Tran, A., Manzanera, A.: A versatile object tracking algorithm combining particle filter and generalised Hough transform. In: 2015 International Conference on Image Processing Theory, Tools and Applications (IPTA). IEEE (2015)

Wu, Yi, Lim, Jongwoo, Yang, Ming-Hsuan: Object tracking benchmark. IEEE Trans. Pattern Anal. Mach. Intell. **37**(9), 1834–1848 (2015)

Yilmaz, A., Javed, O., Shah, M.: Object tracking: a survey. ACM Comput. Surv. (CSUR) **38**(4), 13 (2006)

Ali Sekhavati received his B.S. in Electrical Engineering from Sadjad University of Technology, Iran in 2020. He is mainly interested in doing research on the subject of Machine Vision and Machine Learning.

Najmeh Eghbal received the B.S. and M.S. degrees and also her Ph.D. in electrical engineering from Ferdowsi University of Mashhad, Iran, in 2001, 2004 and 2012, respectively. She is an assistance professor at Sadjad University of Technology. Her main research interests are modelling and control of hybrid systems and machine vision.