Performance evaluation for tracker-level fusion in video tracking

by

ObaidUllah Khalid

Bachelor in Computer Engineering 2004

Master in Network Engineering 2008

A dissertation submitted to

The School of Electronic Engineering and Computer Science

in partial fulfilment of the requirements for the Degree of

Doctor of Philosophy

in the subject of

Electronic Engineering

Queen Mary University of London

Mile End Road

E1 4NS, London, UK

November 2016

QMUL supervisor UNI-KLU supervisor Author

Professor Andrea Cavallaro **Professor Bernhard Rinner** **ObaidUllah Khalid**

Performance evaluation for tracker-level fusion in video tracking

Abstract

Tracker-level fusion for video tracking combines outputs (state estimations) from multiple trackers, to address the shortcomings of individual trackers. Furthermore, performance evaluation of trackers at run time (online) can determine low performing trackers that can be removed from the fusion. This thesis presents a tracker-level fusion framework that performs online tracking performance evaluation for fusion.

We first introduce a method to determine time instants of tracker failure that is divided into two steps. First, we evaluate tracking performance by comparing the distributions of the tracker state and a region around the state. We use Distribution Fields to generate the distributions of both regions and compute a tracking performance score by comparing the distributions using the L_1 distance. Then, we model this score as a time series and employ the Auto Regressive Moving Average method to forecast future values of the performance score. A difference between the original and forecast returns the forecast error signal that we use to detect tracking failure. We test the method with different datasets and then demonstrate its flexibility using tracking results and sequences from the Visual Object Tracking (VOT) challenge.

The second part presents a tracker-level fusion method that combines the outputs of multiple trackers. The method is divided into three steps. First, we group trackers into clusters based on the spatio-temporal pair-wise relationships of their outputs. Then, we evaluate tracking performance based on reverse-time analysis with an adaptive reference frame and define the cluster with trackers that appear to be successfully following the target as the *on-target* cluster. Finally, we fuse the outputs of the trackers in the *on-target* cluster to obtain the final target state. The fusion approach uses standard tracker outputs and can therefore combine various types of trackers. We test the method with several combinations of state-of-the-art trackers, and also compare it with individual trackers and other fusion approaches.

Contents

Acknowledgments

This PhD Thesis has been developed in the framework of, and according to, the rules of the Erasmus Mundus Joint Doctorate on Interactive and Cognitive Environments EMJD ICE [FPA n° 2010-0012] with the cooperation of the following Universities:

Alpen-Adria-Universität Klagenfurt – AAU

Queen Mary, University of London – QMUL

Technische Universiteit Eindhoven – TU/e

Università degli Studi di Genova – UNIGE

Universitat Politècnica de Catalunya – UPC

According to ICE regulations, the Italian PhD title has also been awarded by the Università degli Studi di Genova.

Acknowledgements

I primarily want to thank my Parents. Their constant moral support and motivational advice has helped me to get to the point where I am now. I want to thank my wife for her continual presence and hard work during this phase of my life, for without her it would have been difficult for me to achieve this milestone.

I want to thank my supervisors Professor Andrea Cavallaro and Professor Bernhard Rinner for providing me with extremely useful and unique technical advice during these four years, and especially for their patience. I will also like to thank Dr. Juan Carlos SanMiguel who helped me during the initial stages of my PhD and was always available to provide me with useful advice.

I want to thank all my friends both in the smart cameras group (London) and pervasive computing group (Klagenfurt) who worked with me and also for the unforgettable moments spent together inside and outside the lab.

Finally, I will like to thank the Almighty for all the countless blessings in my life.

This work was supported by the EACEA, under the EMJD ICE Project.

Published work

[J1] O. Khalid, J.C. SanMiguel and A. Cavallaro. Multi-Tracker Partition Fusion. *IEEE Transactions on Circuits and Systems for Video Technology*, 2016. doi: 10.1109/TCSVT.2016.2542699

[C1] O.Khalid, A. Cavallaro and B. Rinner. Detecting tracking errors via forecasting. *Proceedings of British Machine Vision Conference*,York, September 19-22, 2016.

Electronic preprints are available at: http://www.eecs.qmul.ac.uk/staffinfo/andrea/publications.html

List of symbols

$\mathbf{I}$	Video sequence	8
I_t	Image frame at time t	8
T	Total number of frames in the video sequence	8
n	Dimension of x_t^k	8
$[u_t, v_t]$	Target position in target state x_t	8
$[w_t, h_t]$	Target width and height in x_t	8
F	A single tracker	9
$\mathbf{z}_t$	Observation taken by tracker F at time t	9
ς_{t-1}	Target model determined by tracker F at time $t-1$	9
$\mathbf{F}$	Set of trackers	9
F^k	Tracker k	9
K	Total number of trackers	9
x_t^k	Target state estimated by tracker F^k at time t	9
y_t^k	Tracking performance score	9
$f(.)$	Function for evaluating tracking performance	9
γ_x	Information extracted using x_t^k for evaluating tracking performance	9
γ	Corresponding information against which γ_x is compared to determine y_t^k	9
N	Trackers that accurately track the target such that $N \leq K$	9
$g(.)$	Function for fusing tracking outputs	9
x_t^*	Final (fused) target state determined using the equal weighted fusion approach	9
x_t^{**}	Final (fused) target state determined using the performance-weighted fusion approach	9
S_t	Tracker-state region extracted from I_t using x_t^k	42
$\mathbf{B}_t$	Background region extracted from I_t	42
$\vec{\Omega}_{\Delta t_1}$	Average displacement and direction of the tracker computed over Δt_1	42
Δt_1	Sliding temporal window for computing $\vec{\Omega}_{\Delta t_1}$	42

$[\hat{u}_t, \hat{v}_t]$	Estimated position of $\mathbf{B}_t$	42
$[\hat{w}_t, \hat{h}_t]$	Estimated width and height of $\mathbf{B}_t$	42
$\lceil . \rceil$	Ceil operation	42
$d_t(i,j,c,m)$	Distribution of S_t generated using colour based Distribution Fields	43
(i,j)	Pixel location in S_t	43
c	RGB feature channel index	43
m	Layer index	43
M	Total number of layers in $d_{S_t}(i,j,c,m)$	43
κ	Size of each layer	43
$*$	Convolution operation	43
h_{σ_1}	2-D Gaussian kernel with standard deviation σ_1	43
d_t^1	Smoothed DF obtained by convolution of $d_{S_t}(i,j,c,s)$ with h_{σ_1}	43
h_{σ_2}	1-D Gaussian kernel with standard deviation σ_2	43
d_t^2	Final DF of S_t obtained by convolution of $d_{S_t}^1$ with h_{σ_2}	43
b_t^a	Smaller regions of $\mathbf{B}_t$	43
$r_{a,t}^2$	DF of each b_t^a	43
y_t^a	L_1 distance between $d_{S_t}^2(i,j,c,m)$ and $d_{b_t^a}^2(i,j,c,m)$	44
α_c	Weight assigned to each channel c	44
$\mu_{S_t}^c$	Mean R, G, B values for S_t	44
$\mu_{b_t^a}^c$	Mean R, G, B values for b_t^a	44
$\mathbf{Y}$	Time series of y_t	45
(P,Q)	ARMA orders	45
$\phi(p)$	AR polynomial used in the ARMA model	47
$\vartheta(q)$	MA polynomial used in ARMA model	47
β	Constant used in the ARMA model	47
Δt_2	Sliding temporal window over which the past values of y_t are use to build the Auto Regressive and Moving Average (ARMA) model	47
ε_t	Estimation errors (residuals) computed as a difference between original (y_t) and estimated time series $(\hat{y}_t)$	47
$\hat{y}_t$	Estimated value of y_t obtained using the estimated parameters and ARMA model at time t	47

$\hat{\phi}(p)$	Estimated AR polynomial used in the ARMA model	47
$\hat{\vartheta}(q)$	Estimated MA polynomial used in ARMA model	47
$\hat{\beta}$	Estimated constant used in ARMA model	47
l	Forecast lead time (length)	47
$\hat{y}_{t+l}$	Forecast time series over the forecast length l	47
$\tilde{e}_{t+l}$	Forecast error computed as a difference between original (y_t) and forecast time series ($\hat{y}_{t+l}$)	47
τ_1	Threshold for detecting tracking failure	47
e_{t+l}	Binary values of $\tilde{e}_{t+l}$ based on τ_1 generated for tracker failure detection	47
δ_t^e	Tracking failure decision generated by the method, measured as $\lvert e_{t+l} - e_{t+l-1}\rvert$	47
O_t	Tracking error measured as deviation from ground-truth (ideal) data	49
A_t^x	Area in pixels of estimated (x_t) target location	49
A_t^{GT}	Area in pixels of ideal target location	49
δ_t^O	Tracking failure decision generated using GT data, measured as $\lvert O'_t - O'_{t-1}\rvert$	49
O'_t	Binary values of O_t generate for tracker failure detection	49
n_{TP}	Number of true positives (TP)	49
n_{FP}	Number of false positives (FP)	49
n_{TN}	Number of true negatives (TN)	49
n_{FN}	Number of false negatives (FN)	49
P	Precision calculated as the ratio between n_{TP} and $(n_{TP} + n_{FP})$	49
R	Recall calculated as the ratio between n_{TP} and $(n_{TP} + n_{FN})$	49
F	F-score	49
FPR	False positive rate calculated as the ratio between n_{FP} and $(n_{FP} + n_{TN})$	49
z_t	Generic variable used for representing $\lvert\tilde{e}_{t+l}\rvert$, y_t and the tracking performance scores generated by the SOA methods	50
$\mathbf{Z}$	The complete time series representing z_t	50
z'_t	Normalized values of z_t between [0,1]	50
Δt_3	Sliding temporal window for buffering data from future time instants	59
Δt_4	Sliding temporal window for buffering data from past time instants	59
$O_t^{i,j}$	Spatial area overlap of trackers F^i and F^j computed using their outputs x_t^i and x_t^j, respectively	60

A_t^i	Area in pixels of the bounding box generated by tracker F^i	60
A_t^j	Area in pixels of the bounding box generated by tracker F^k	60
$O_{\Delta t_3}^{i,j}$	Average spatial agreement score over Δt_3 between trackers F^i and F^j	60
$\vec{\mathbf{d}}_t^k$	Directional feature of each F^k	60
$r_{\Delta t_3}^{i,j}$	Directional similarity score over Δt_3 between trackers F^i and F^j	60
$\psi(\lambda, r_{\Delta t_3}^{i,j})$	Weighting function for normalizing $r_{\Delta t_3}^{i,j}$ between $[0,1]$	61
$\hat{r}_{\Delta t_3}^{i,j}$	Weighted directional similarity score	61
λ	Decay rate of weighting function $\psi(\lambda, r_{\Delta t_3}^{i,j})$	61
$R_{\Delta t_1}^{i,j}$	Spatio-temporal pair-wise correlation score between trackers F^i and F^j	62
ω	Weight to prioritize between $\tilde{O}_{\Delta t_3}^{i,j}$ and $r_{\Delta t_3}^{i,j}$ in $R_{\Delta t_1}^{i,j}$	62
$\mathbb{P}_{p,t}$	Single partition of $\mathbf{F}$ where $\lvert\mathbb{P}_{p,t}\rvert \in [1,K]$	62
p	Partition index	62
$\mathbb{C}_{p,t}^a$	Non-empty mutually disjoint cluster of trackers $\in \mathbb{P}_{p,t}$	62
a	Cluster index	62
$H(\xi^*(p))$	HC based function that provides $\mathbb{P}_{p,t}$ using optimum distance threshold β^*	63
$\xi^*(p)$	Optimum (smallest) distance threshold value for selecting partition p	63
ξ	A set $\in [0,(1-R_{\Delta t_1}^{i,j})]$ containing $R_{\Delta t_1}^{i,j}$ scores for all tracker pairs	63
$S(\mathbb{P}_{p,t})$	Score of a partition $\mathbb{P}_{p,t}$	63
$Q\left(\mathbb{C}_{p,t}^a\right)$	Score of a cluster $\mathbb{C}_{p,t}^a$	64
$\mathbb{P}_t^*$	Selected partition at time t based on maximum partition score $S(\mathbb{P}_{p,t})$	64
ν	Total number of tracker-pair combinations	64
$\mathbb{C}_t^*$	Selected *on-target* cluster at time t based on reverse-evaluation	65
l_t^k	Label $\in \{$*on-target*, *off-target*$\}$ assigned to each F^k at time t	66
$x_t^{k,-}$	Target state estimated by reverse tracker k at frame t	66
I_{ref}	Reference frame where results from forward and reverse tracker are compared	66
$G(x_{ref}^{k,-}, x_{ref}^*)$	Function for comparing results of the reverse tracker with the fused output	67
x_{ref}^*	State estimation obtained from the fused output at I_{ref}	67
$x_{ref}^{k,-}$	State estimation obtained from the reverse tracker at I_{ref}	67
θ_t^k	Similarity score between x_{ref}^* and $x_{ref}^{k,-}$, used for evaluating performance of F^k	67
$A_{ref}^{x^*}$	Area in pixels of the bounding box generated by the fused output at I_{ref}	67
$A_{ref}^{k,-}$	Area in pixels of the bounding box generated by the reverse tracker at I_{ref}	67

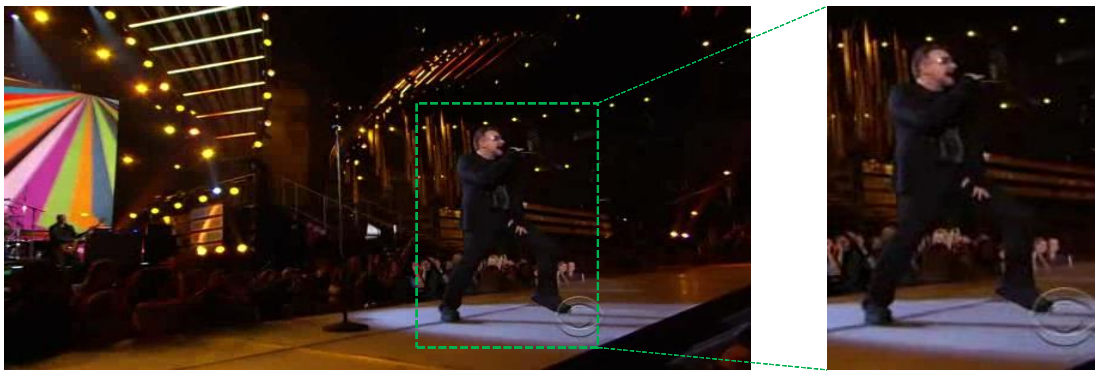
$M_{u_1^k}$	Motion of the coordinate u_1 for tracker F^k computed over the temporal window $deltat_4$	67
M^k	Motion of tracker F^k taken as the maximum of the motions of four individual coordinates, top-left $(u_{1,t}^k, v_{1,t}^k)$ and bottom-right $(u_{2,t}^k, v_{2,t}^k)$	68
M^{x^*}	Motion of fused output x_t^* computed over the temporal window $deltat_4$	68
τ_2	Threshold for comparing performance (θ_t^k) of trackers	68
$\check{y}_t^n$	Normalized weight of each tracker	69
$O_t^{\mathbb{C}^a}$	Overlap score for cluster $\mathbb{C}^a$ measured as the average of $(1-O_t)$ of all trackers in $\mathbb{C}^a$	71
τ_3	Threshold for comparing $(1-O_t)$ and $O_t^{\mathbb{C}^a}$ to define *on-target* trackers and cluster, respectively	71
O_G	Mean overlap score taken as the mean over the complete sequence	71

Chapter 1

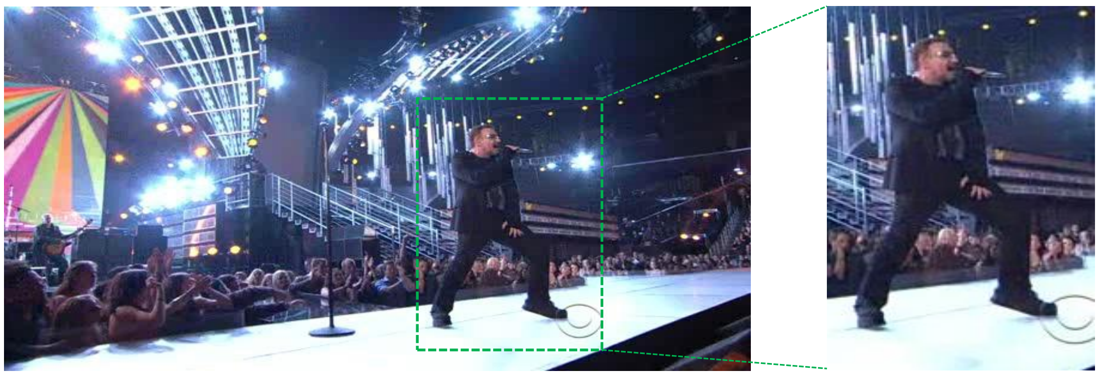
Introduction

1.1 Motivation

Visual tracking within a single-target tracking framework refers to estimating the state of the target over time (*trajectory*) in a video sequence, by achieving a congruence between a pre-defined model of the target and measurements at each time step. The state of the target can either be the position, shape, area, size, velocity, orientation or a combination of these properties of the target. It has become an important component within the field of computer vision and is widely used in numerous applications such as video surveillance, traffic monitoring, human computer interaction, video indexing, object-based video compression, motion analysis and activity recognition over the past years. However, the video data used pose numerous challenges such as changes in appearance, illumination (Figure 1.1), scale (Figure 1.2), velocity and orientation (Figure 1.3) of the target, partial or full occlusions (Figure 1.4) and background clutter (Figure 1.5). Moreover, challenges such as motion blur, sudden changes in target motion and low quality camera sensors (e.g. low frame rate or low resolution) further increases the complexity of the data analysed. A wide variety of tracking algorithms such as methods based on the Kalman Filter (KF) [66], Particle Filter (PF) [109] and the Mean Shift (MS) [32] have been proposed to tackle the different challenges. Despite these recent developments, no single method can provide flawless results for all challenges [125]. For example a tracker might perform better under occlusions, it might lose fast moving objects. On the other hand another tracker might perfectly handle changes in illumination but the presence of similar coloured background might cause the tracker to fail. Tracking



(a)

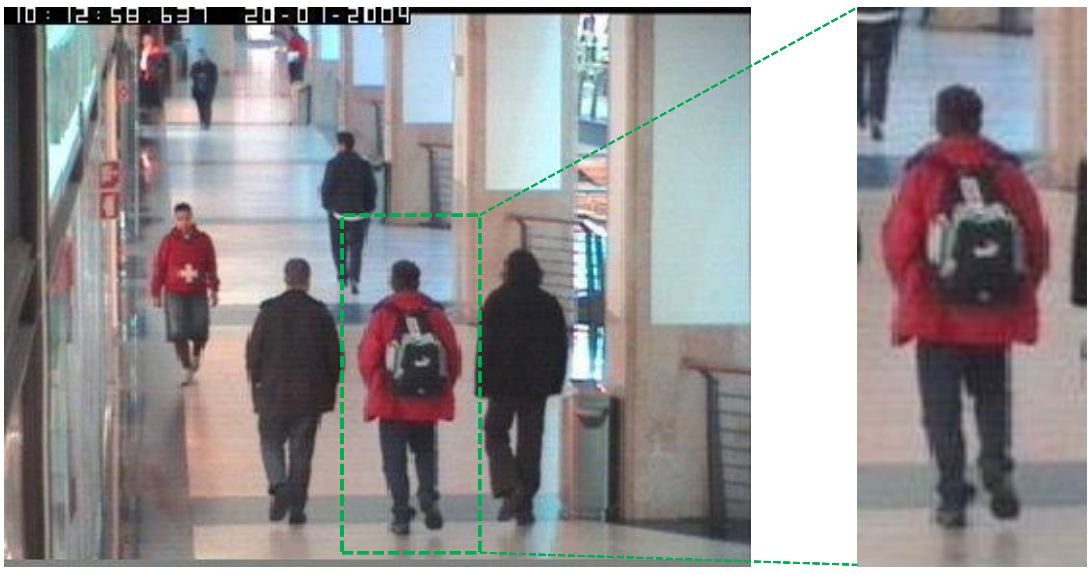


(b)

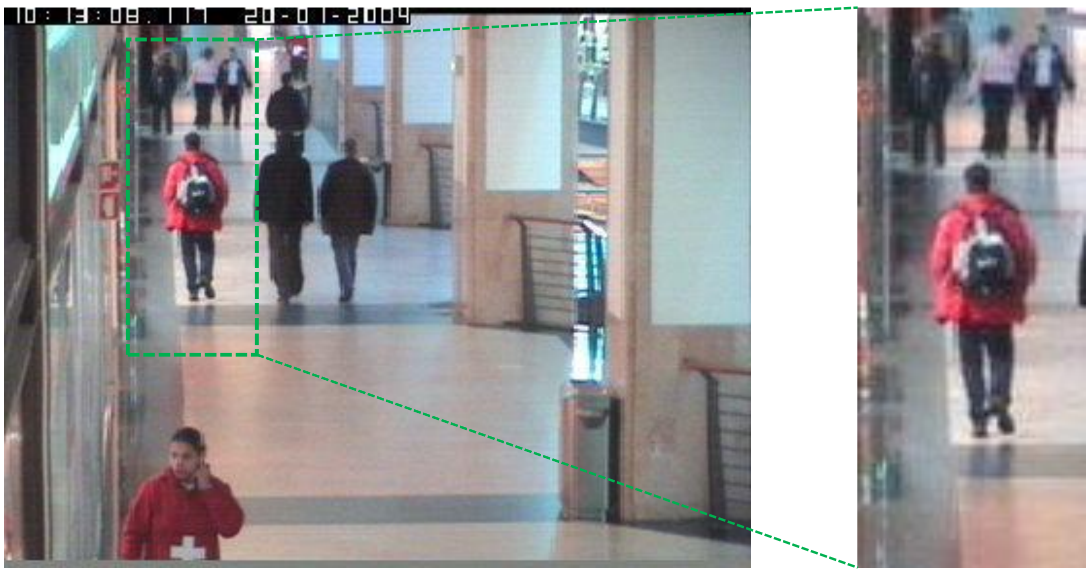
Figure 1.1: Example of colour variations of the target due to illumination variations: (a) low background lightning, (b) strong background lightning.

failure can be defined as target loss during tracking (drifts away from the target) [72], and can be quantified when the amount of area overlap between the estimated and Ground Truth (GT) (ideal) position of the target is zero [72, 114, C1]. Automatic recovery from a tracking failure is difficult and most tracking algorithms do not explicitly detect tracking failures.

One possible solution to the problem can be combining different visual trackers within a fusion framework where each method has a complementary failure mode. Fusion in visual tracking enables combining information from multiple sources (features or trackers) to improve the overall tracking performance. This fusion may occur either at *feature-level* or at *tracker-level*. *Feature-level* fusion methods combine multiple features within a single tracking framework to create robust target representations [7, 43, 68, 91, 92, 94, 127, 135, 151, 153, 162]. However, due to variable dimensionality and ranges of different features, adaptation methods are needed to integrate new features [76]. *Tracker-level* fusion methods implicitly combine features by fusing the outputs of multiple trackers [16, 49, 59, 74, 76, 118, 168, J1]. *Tracker-level* fusion can either be performed in a sequential manner [96, 118, 124] where the output of a tracker is fed as input to the



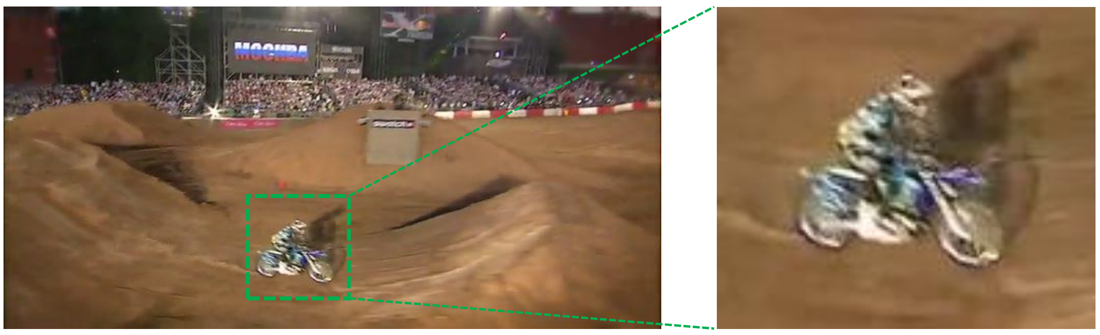
(a)



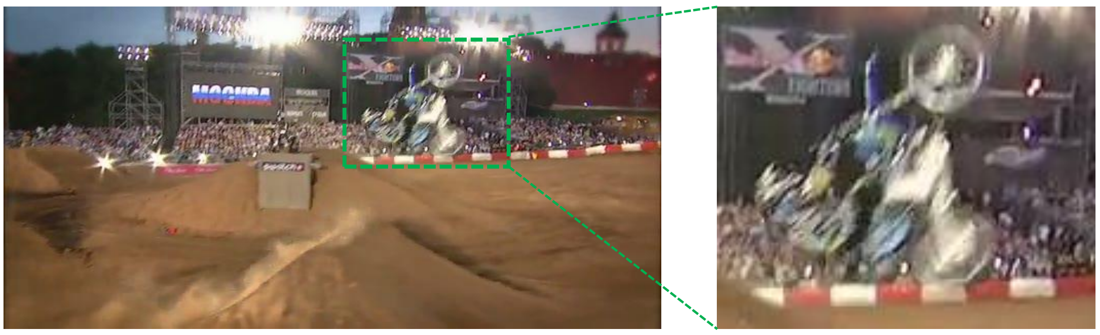
(b)

Figure 1.2: Example of scale changes due to the target moving away from the field of view of camera: (a) target area at closer distance, (b) target area at farther distance.

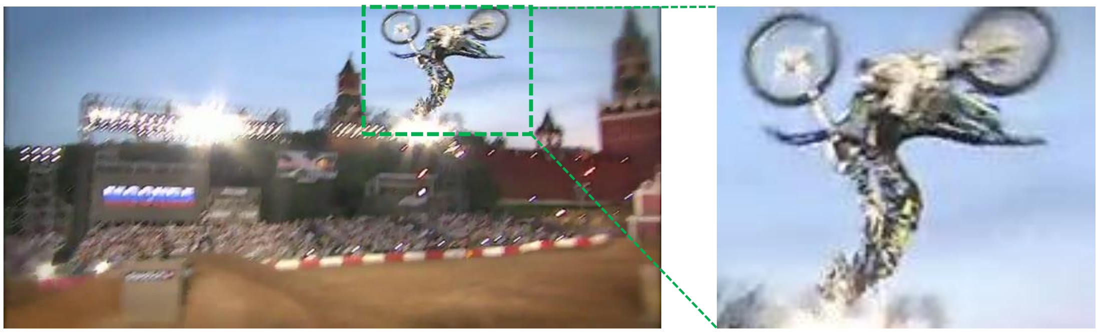
next tracker, in parallel [16,49,J1] where each tracker generates an individual output, by combining outputs of specific trackers [89, 168] or by employing trackers that work in collaboration as a single tracker [74]. However, a blind fusion scheme may not help in improving the overall results as failing or low performing trackers may still reduce the tracking accuracy. An online performance evaluation method that determines the accuracy of the features [91] or trackers [116] at run time can be employed for performance weighting of the features [91] or trackers [49] within the fusion framework. Such methods can be used to correct [16] or remove [J1] low performing



(a)



(b)

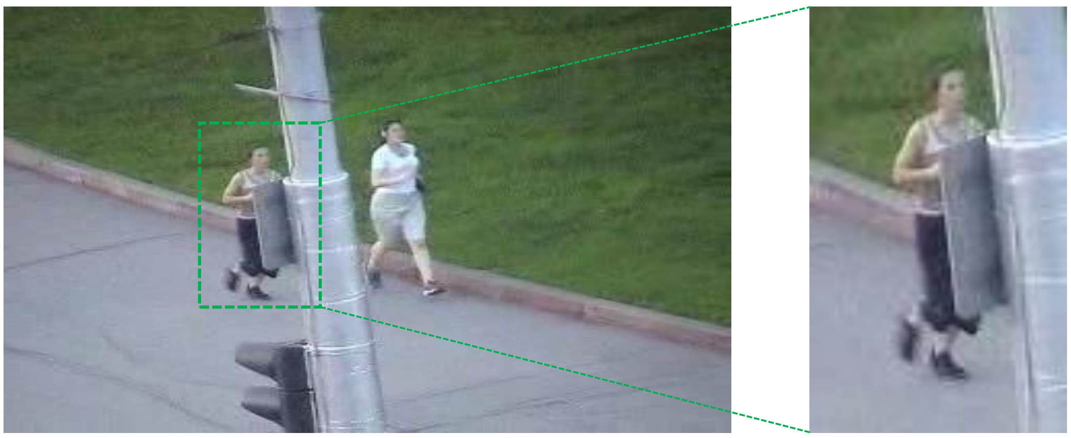


(c)

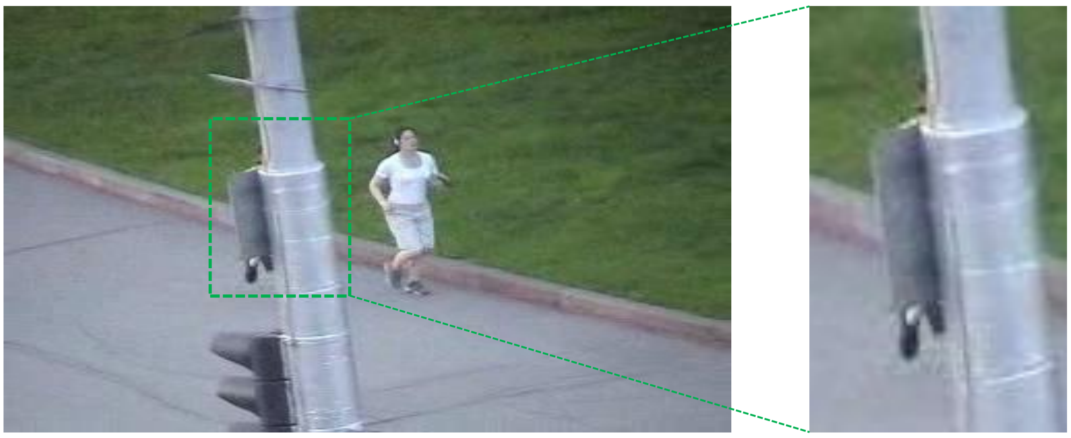
Figure 1.3: Example of changes in target orientation due to rotations at: (a) 0°, (b) 90°, (c) 180°

trackers within the fusion framework.

Performance evaluation of a tracker refers to determining the tracking performance using either an offline (GT based) or an online (GT free) technique. This performance can be quantified as the accuracy i.e. the closeness of the estimated state to the ideal state [90] or the robustness i.e. the number of times the tracker fails during tracking [73]. Offline performance evaluation methods quantify the accuracy [77,125,154], the robustness [72] or a combination of both [98] by measuring the error between the estimated state and GT data. Common measures of quantifying this error include the distance between the estimated and GT positions [125] or the amount of overlap between the estimated and GT states [154].



(a)



(b)

Figure 1.4: Example of an occluded target due to foreground objects: (a) partial occlusion, (b) full occlusion.

Online performance evaluation methods estimate tracking accuracy [116,J1] or robustness [113, 114, C1] without using GT data and attempt to identify the *on-target* (successfully following the target) trackers using current and past information only [155]. These methods can make use of measures that are based on *features*, *trajectories* or a combination of both (*hybrid*). *Feature* based measures that are dependent on specific trackers exploit the internal properties of the trackers such as the observation likelihood [74] or the spatial uncertainty of PF [16, 116, 151]. Feature descriptors independent of the tracker are determined using the tracker output and estimate tracking performance by temporal comparisons of the descriptors [113] or by quantifying the ability of the descriptor to discriminate the state from the surrounding background [C1]. *Trajectory* based measures use tracker outputs to evaluate tracking performance. This can be achieved by measuring the consistency of target velocity [123], comparing outputs of the tracker and the same tracker running in time-reversed direction [149, J1] or by measuring the spatial closeness of the

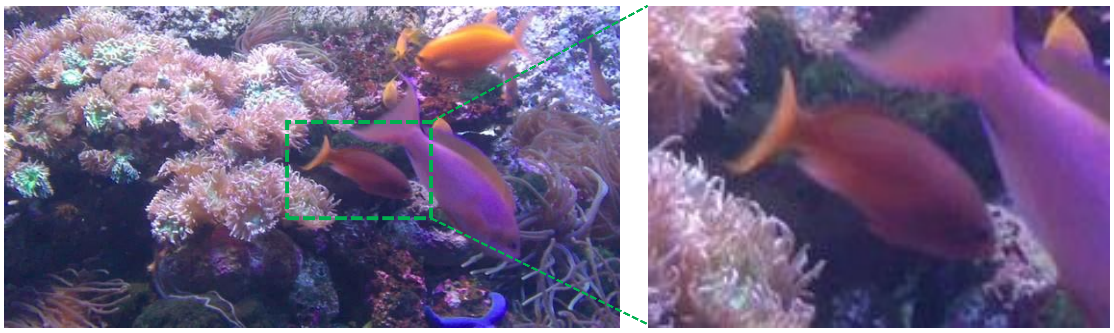
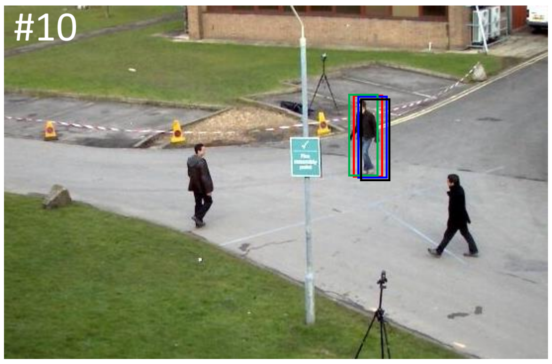


Figure 1.5: Example of background clutter where the target to be tracked is not clearly distinguishable due to similar coloured background or other similar objects.

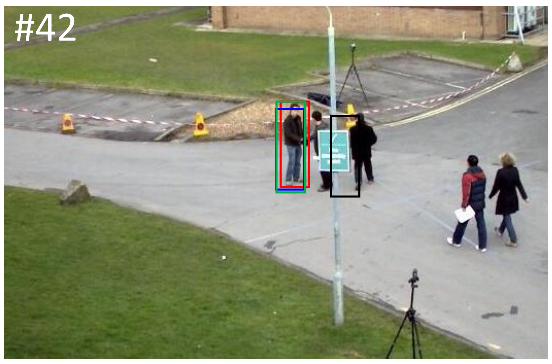
estimated states of multiple trackers in a fusion framework [49]. *Hybrid* approaches estimate tracking accuracy by combining both feature and trajectory based evaluation measures [26, 126]. These evaluation measures can then be exploited to determine the time instants of tracker failures. This can be accomplished by analyzing the temporal changes of the spatial uncertainty of PF [114, 116] or by employing thresholds over the tracking accuracy score [113, 149, C1].

The primary objective of a *tracker-level* fusion framework is to combine the strengths of multiple trackers that have different failure modes. All trackers are initialized on the same region of the image (target) at the first frame as a single group (cluster). We define a cluster as a group of trackers having state estimations within the same spatial region. However, due to the different challenges present in the visual data, these trackers fail at different time instants and drift away from each other. This results in the trackers forming multiple clusters at each time step. An example tracking scenario with four trackers is presented in Figure 1.6. Using the outputs of the trackers, these clusters can be identified by quantifying the correlation between pairs of trackers, where a correlation can be defined as the spatial agreement between tracking outputs [49] or a combination of both spatial and temporal agreements between tracking outputs [J1]. An online performance evaluation method can be used to identify the cluster that is *on-target* i.e. correctly following the target, while the remaining trackers can be discarded. The final target state can then be determined by combining outputs from trackers within the *on-target* cluster. Current State-Of-the-Art (SOA) approaches do not attempt to identify these clusters, and either use all trackers within performance evaluation and fusion [16, 49] or select the one with the highest performance score [74]. Using all trackers at each time instant results in redundant computations, while selecting a single tracker may discard other trackers that are still *on-target*.

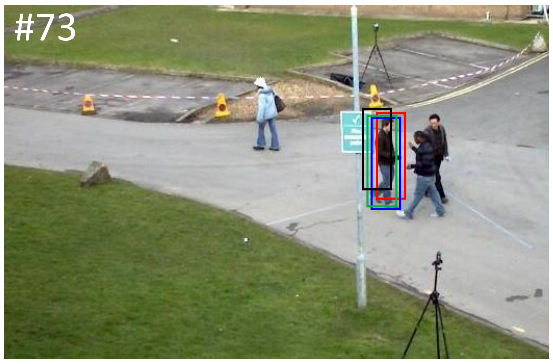
In this thesis, we present a parallel framework for tracker-level fusion that employs two separate online performance evaluation methods. In the first part of this work, we present an online performance evaluation method that estimates tracking accuracy at each time instant using feature descriptors that are generated using outputs from the tracker only. The tracking accuracy is quantified as the ability of the feature descriptors to discriminate the state from the surrounding background region. Then, we employ a forecasting technique to determine time instants of tracking failure by applying a threshold over the accuracy score. We test the method using two diverse datasets and compare it against SOA tracking failure detection methods. Since the method uses the outputs of a tracker (e.g. bounding boxes) only, it has been proposed for performance evaluation of generic individual trackers. In the second part of this work, we make use of the spatio-temporal correlation between pairs of trackers to determine the clusters of trackers. We measure the spatial correlation as the amount of overlap between the trackers' outputs, while the temporal correlation is quantified as the directional similarity within a short temporal window. We then make use of the time-reversed based performance evaluation method to determine the accuracy of each individual tracker and select the *on-target* cluster. The time-reversed evaluation method [149] however, imposes a heavy computational cost due to the requirement of running the tracker in the reverse-direction across all the frames. To reduce the computational complexity we propose an improvement to the time-reversed evaluation method by determining a stopping criteria for the reverse analysis (the reference frame) which is adaptively determined and updated. The performance evaluation of the trackers (at the reference frame) helps in the identification of the *on-target* cluster that is propagated in time until the cluster changes. A change in the cluster is defined as a split (i.e. tracker(s) leaving the cluster) or a merge (i.e. tracker(s) joining the cluster). The propagation of the same cluster over time reduces the number of additional computations that would be required for evaluating the performance of each tracker at each time step. The final target state is then estimated by combining the outputs from the trackers within the selected *on-target* cluster. Fusion is performed using the equal weighted and performance-based weighted approach, where the tracking accuracy computed in the first part of the work is used as the performance weight of each individual tracker. We test the fusion framework with different combinations of multiple SOA trackers and compare it with SOA fusion approaches.



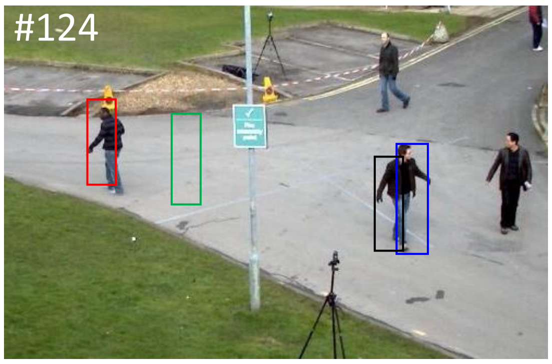
(a)



(b)



(c)



(d)

Figure 1.6: The image illustrates the split and merge of trackers creating new clusters over time in a video sequence. —: Tracker1; —: Tracker2; —: Tracker3; —: Tracker4. (a) Frame #10: All trackers are part of a single cluster. (b) Frame #42: The trackers split into two clusters, when Tracker4 fails due to an occlusion. (c) Frame #73: Tracker4 re-acquires the target and all four trackers merge again into a single cluster. (d) Frame #124: Over time, the four trackers split into three different clusters, where only Tracker3 and Tracker4 remain *on-target*.

1.2 Problem definition

Let $\mathbf{I} = \{I_t\}_{t=1}^{T}$ be a video sequence, where I_t is the frame at time t and T is the total number of frames. Visual tracking refers to estimating the target state over time, where $x_t \in \mathbb{R}^{\mathsf{n}}$ defines the target state and n represents the dimension of the state. x_t can either include the position, velocity and orientation [103] or the position and shape [42] of the target. Using $\mathsf{n} = 4$ for example, the target state can be defined as a bounding box i.e. by its 2-D position and size as:

$$x_t = [u_t, v_t, w_t, h_t], \tag{1.1}$$

where u_t, v_t are the target position with respect to the horizontal and vertical axes, respectively and w_t and h_t are its width and height, respectively. Furthermore, x_t over time, $\{x_t : 1 \leq t \leq T\}$

is the trajectory of the target. Let $F(.)$ define the tracker that estimates x_t over time as:

$$x_t = F(x_{t-1}, \mathbf{z}_t, \varsigma_{t-1}), \tag{1.2}$$

where $\mathbf{z}_t$ is the measurement at time t, x_{t-1} and ς_{t-1} is the estimated state and the target model, respectively at time $t-1$.

The objective of tracker-level fusion in video tracking is to combine outputs (state) of multiple trackers that may enable to overcome the limitations of each individual tracker. Let $\mathbf{F} = \{F^k\}_{k=1}^{K}$ be a set of K trackers, where F^k is the k^{th} tracker and the target state estimated by tracker F^k is represented as x_t^k. It is desirable to work with standard tracking outputs such as a bounding box [42, 143, 166] or a bounding ellipse [102, 103] to make the framework generic towards all type of trackers. Using x_t^k the performance of the tracker at run time can be evaluated as:

$$y_t^k = f(\gamma_x, \gamma), \tag{1.3}$$

where $f(.)$ defines the performance evaluation function, y_t^k is the tracking performance score for F^k, γ_x is the information extracted using x_t^k and γ is the corresponding information against which γ_x is compared to determine y_t^k. γ_x may be derived from trajectories [49, J1] or features extracted from I_t using x_t [113, C1]. Similarly, γ can be extracted from additional trajectories generated by other trackers [149], background features [C1] or the same features extracted at the previous time step, $t-1$ [113]. Values for y_t^k are dependent on the information (trajectories or features) used to derive γ_x and γ, where y_t^k may have ranges $\in [0, 1]$ [49, J1] or $\in [0, \infty]$ [116, 149] indicating the confidence of F^k in tracking the target.

Such performance evaluation can enable to estimate failed trackers i.e. the ones having an inaccurate estimation of the target state. Failing trackers can be discarded and the outputs from the remaining N (such that $N \leq K$) trackers can be combined to estimate the final target state as:

$$x_t^* = g(\{x_t^n\}_{n=1}^{N}), \tag{1.4}$$

or

$$x_t^{**} = g(\{x_t^n\}_{n=1}^{N}, y_t^k), \tag{1.5}$$

where $g(.)$ is the fusion function and x_t^* or x_t^{**} is the final target state determined either by an equal weighted or a performance-based weighted fusion approach, respectively.

1.3 Contributions

This thesis presents a parallel framework for combining the outputs of multiple trackers to improve the overall tracking accuracy. The first part introduces an online performance evaluation method that is used to determine tracking failure and for performance weighting of the trackers during fusion. While in the second part of this work we introduce a clustering based approach for tracker-level fusion that estimates the final target state by combining outputs from *on-target* trackers only. The detailed contributions of the framework are:

1. *State-background discrimination approach for tracking accuracy estimation*. We provide a generic approach for estimating tracking accuracy. Since the approach uses standard tracking outputs, it can be applied to any kind of tracker. The approach compares the regions defined by the tracker state and the background region around the state to estimate tracking accuracy over time. Using motion information from short-term trajectories, the method first computes and extracts a larger background region around the target state. Then using Distribution Fields (DF) [122], distributions of both the state and background region are generated, which are then compared using the L_1 distance to estimate tracking accuracy [C1].

2. *Tracking failure detection via forecasting*. To determine when the tracker fails, we apply a forecasting approach on the track accuracy score (obtained from the state-background discrimination approach). We select Auto Regressive Moving Average (ARMA) [22] as the forecasting model, that predicts future values of the track accuracy score using current and past information only. Then the difference between the original and forecast values generates the forecasting error signal. Tracking failures are highlighted within the forecast error signal as significant changes which are detected using an experimentally derived threshold [C1].

3. *A clustering approach to tracker level fusion*. We introduce a clustering approach to identify different groups (clusters) of trackers within the tracker-level fusion framework [J1]. We determine pair-wise spatio-temporal correlation between trackers based on their estimated states and then employ the hierarchical clustering-based approach to determine the possible

clusters. This allows us to determine different clusters of trackers at each time instant and then only use the cluster that is on the correct target. Existing methods to tracker-level fusion [16, 49, 74] do not exploit this information and instead work with all the trackers at each time instant.

4. *An improved reverse-analysis method to online tracker evaluation*. After determining the clusters, we employ an online performance evaluation method to determine the *on-target* cluster. The performance evaluation method is based on the time-reversed analysis approach that estimates tracking accuracy by comparing results of the tracker with the results of the same tracker running in the reverse time direction [149]. This comparison is performed at a reference frame (first frame of the sequence where the target is initialized using GT data [149]). However, the constraint of running the tracker in reverse direction up until the first frame of the sequence imposes a heavy computational cost, especially when it is employed for online evaluation of the tracker. We adaptively update and move the reference frame forward in time by employing the motion information of the tracker trajectory. Furthermore, to determine tracking accuracy at the reference frame, we compare the results of the reverse-tracker to the fused output at the reference frame [J1]. The performance evaluation method enables us to identify the *on-target* trackers and hence the *on-target* cluster.

5. *Fusion of selected trackers*. Existing tracker-level fusion methods estimate tracking accuracy for assigning performance-based weights to both *on-target* and failing trackers. And the final target state is estimated by either combining outputs from all trackers [16,49] or by selecting the tracker with the highest weight [74]. The strategy of combining outputs from all trackers can however lower the overall tracking accuracy and the approach to using only the best tracker can be a drawback where other trackers that can be *on-target* are discarded. We propose an approach to fusion that combines outputs from the trackers only within the *on-target* cluster, while the remaining (failing) trackers are discarded.

1.4 Organisation of thesis

The thesis is organised as follows:

Chapter 1: Introduction and motivation for this work, the problem formulation and the contributions of this thesis.

Chapter 2: This chapter covers related works of visual tracking with definitions of different types of visual tracking methods. Discussions on both the offline and online methods used for performance evaluation of visual trackers. Definitions of methods used for fusion in visual tracking and discussions on feature and tracker-level fusion techniques. Finally, a discussion on the limitations of the fusion and online performance evaluation methods concludes the chapter.

Chapter 3: An online performance evaluation technique used for evaluating tracking performance and for determining the time instants when the tracker fails. Experimental analysis and comparison with the state-of-the-art methods. The work in this chapter appears in [C1].

Chapter 4: The clustering approach used for determining the clusters of trackers using the relationships between tracking outputs. An online performance evaluation method based on the adaptive time-reversed evaluation approach for determining the *on-target* cluster. Tracker fusion method for combining the outputs of the trackers within the *on-target* cluster. The experimental setup that is followed by experimental validation of each of the three parts: the tracker clustering, adaptive time-reversed evaluation and tracker fusion. Experimental analysis of the computational cost of the complete method. The work in this chapter appears in [J1].

Chapter 5: Summary of achievements and future research directions.

Chapter 2

State of the art

2.1 Introduction

Validating the quality of a tracker over time can help in determining when the performance of the tracker declines. Estimating this quality at run time (without using any GT data) requires an online performance evaluator [116, 149] that can further be used to detect time instants where the tracker fails [C1]. Tracking performance evaluation can then be employed within a fusion framework where fusion involves combining multiple components (features or trackers) to improve the over all tracking performance. This evaluation can help in either ranking [31], correcting [15] or removing [J1] poor performing components during the fusion process. Fusion for visual tracking can occur either at feature-level [68, 151] or at tracker-level [49, J1], where feature-level fusion methods combine multiple features within a single framework, while tracker-level fusion methods combine the outputs generated by multiple tracking algorithms.

This chapter provides a review of the SOA in the field of visual tracking including methods for performance evaluation and fusion within the visual tracking framework. A review on visual tracking is covered in Section 2.2 and Section 2.3 covers the related work in offline performance evaluation. Related work on online performance evaluation and tracking failure detection are covered in Section 2.4. Section 2.5 covers the techniques of fusion for visual tracking, divided into feature-level and tracker-level fusion. Finally, Section 2.6 provides a discussion on the state-of-the-art and a comparison with our framework.

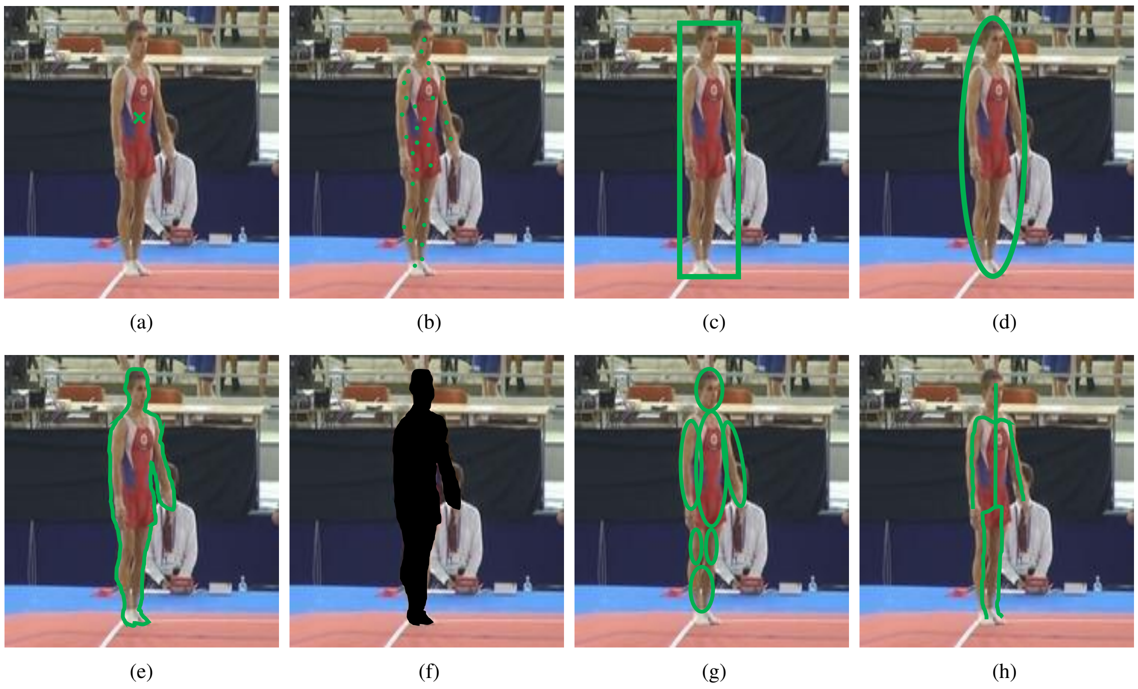


Figure 2.1: Representation of the state of a target. (a) center point, (b) set of points, (c) bounding box, (d) bounding ellipse, (e) contour, (f) silhouette, (g) part-based multiple patches, (h) skeleton. Cropped image is taken from the *gymnastics* sequence of the VOT2014 dataset (Appendix A). This figure is an adaptation of Figure 1 from [158].

2.2 Visual tracking

In general methods of visual tracking primarily differ based on target representation (shape), the image features used, the appearance and the motion model, and the tracking environment. Moreover, tracking methods can be categorized either into deterministic and probabilistic [171] or into generative and discriminative [166] as explained below.

2.2.1 Target representation

A target can either be represented as a point (center position) or set of points [47], bounding box [104], bounding ellipse [32, 103], contour, silhouette, part-based multiple patches [2, 42] or object skeleton (see Figure 2.1), while its appearance is modelled by using features extracted from the image.

Feature selection plays an important role in visual tracking, where an important property of a visual feature is its ability to discriminate the target from the surrounding background. Low-level features include colour (intensity values) [103, 135], gradient (difference between intensity values in one frame) [13] and motion-patterns (changes in intensity values over time) [121]. Colour or intensity features are directly obtained from the pixels of the image and their low computational cost makes them a desirable feature to exploit. Colour-based appearance models such as colour histograms [32, 103] are robust to partial-occlusions and scale changes. However, colour-based appearance models are sensitive to illumination changes, background clutter and can be mis-lead by objects with similar colour properties [91]. In comparison to colour features, gradient features are less sensitive to illumination variations and are primarily applied for edge detection [158]. Motion pattern-based methods such as optical flow [121] record the translation of each pixel within a region and are used for motion-based segmentation and tracking applications. Combining multiple low-level features such as colour and gradient features [13, 91], colour and interest points [47], colour and position [113] or colour and optical-flow [64] generates medium-level features and makes the combination of the advantages of each individual low-level feature possible. The complete region or the orientation of the target are termed as high-level features that can be generated by a combination of medium-level features [90].

The motion of the target over time is defined by a motion model that helps in predicting the likely target position in the next frame. This prediction can be achieved by using different models such as constant velocity [91], adaptive velocity [171], constant acceleration [163], Gaussian [110] and affine motion [80, 143]. Models based on the constant velocity or acceleration can help in tracking occluded targets while the random walk model can be employed for targets that do not have a predefined motion pattern. Employing models such as the affine motion model that includes the position, scale, rotation, aspect ratio and the skew angle of the target bounding box [143] can increase robustness to target rotations and changes in scale.

2.2.2 Deterministic and probabilistic tracking methods

Deterministic tracking methods usually look to minimize a cost function e.g. the Sum of Squared Distance (SSD) [52] or the Bhattacharyya distance in a MS tracking framework [32, 102, 140]. The MS tracking framework, is one of the most commonly used deterministic approach that works by minimizing the cost function (distance between the reference and the target model) and tracks by searching in the neighbourhood of the target's estimated position from the previous frame. The standard MS tracker [32] models the target appearance using an m-bin histogram in the colour (RGB) feature space:

$$\hat{\mathbf{q}} = \{q_e\}_{e=1,\dots,m}; \sum_{e=1}^{m} q_e = 1, \tag{2.1}$$

where q_e is the probability of feature $e = 1, ..., m$ and is determined as:

$$q_e = C \sum_{i=1}^{n} k\left(\|[u'_i, v'_i]\|^2\right) \delta[b([u'_i, v'_i]) - e], \tag{2.2}$$

where $[u'_i, v'_i]$ are the pixel locations with total n pixels, $k(.)$ is a Gaussian kernel, the function $b : R^m \rightarrow 1, ..., m$ maps the value of pixel at location $[u'_i, v'_i]$ to index $b([u'_i, v'_i])$ of the corresponding bin in the feature space, δ is the Kronecker delta and C is the normalization constant i.e. $\sum_{e=1}^{m} q_e = 1$. A target candidate at location $[u, v]$ in the next frame is described by its histogram:

$$\hat{\mathbf{p}} = \{p_e([u, v])\}_{e=1,...,m}; \sum_{e=1}^{m} p_e = 1, \tag{2.3}$$

and the probability of feature e in the target candidate is defined using the same kernel profile $k(.)$ but with a scale parameter h as:

$$p_e([u, v]) = C_h \sum_{i=1}^{n_h} k\left(\left\|\frac{[u, v] - [u_i, v_i]}{h}\right\|^2\right) \delta[b([u_i, v_i]) - e], \tag{2.4}$$

where $[u_i, v_i]$ are the pixel locations with total n_h pixels and C_h is the normalization constant. The distance between the two distributions is determined as:

$$H(\hat{\mathbf{p}}, \hat{\mathbf{q}}) = \sqrt{1 - \rho[\hat{\mathbf{p}}, \hat{\mathbf{q}}]}, \tag{2.5}$$

where

$$\rho[\hat{\mathbf{p}}, \hat{\mathbf{q}}] = \sum_{e=1}^{m} \sqrt{p_e([u, v]) q_e}, \tag{2.6}$$

is the Bhattacharyya coefficient. The search for the target's new location in the current frame starts from the target's position in the previous frame $[u_0, v_0]$ within a fixed-shape variable size window. This search is based on the MS method that works to find the mode of the function i.e. the one that maximizes the Bhattacharyya distance. The kernel is moved from the current location $[u_0, v_0]$ to the new location $[u_1, v_1]$ as:

$$[u_1, v_1] = \frac{\sum_{i=1}^{n_h} [u_i, v_i] w_i g\left(\left\|\frac{[u_0, v_0] - [u_i, v_i]}{h}\right\|^2\right)}{\sum_{i=1}^{n_h} w_i g\left(\left\|\frac{[u_0, v_0] - [u_i, v_i]}{h}\right\|^2\right)}, \tag{2.7}$$

where

$$w_i = \sum_{e=1}^{m} \sqrt{\frac{q_e}{p_e([u,v])}} \delta[b([u_i,v_i]) - e], \tag{2.8}$$

is the weight and $g(.) = -k'(.)$ is the derivative of $k(.)$. A main advantage of the MS-based trackers are their low computational complexity, however they can fail to track occluded or fast moving targets.

Probabilistic (stochastic) methods formulate the process of target tracking as an estimation problem i.e. estimating the state of the target over time using both the appearance and motion models. The KF [66], the PF [16, 91, 103, 109, 171] and their variants [4] are the most widely used approaches to probabilistic tracking. However KF-based tracking algorithms are limited by their assumption of a Gaussian distribution and linear model for the target state and motion, respectively. PFs also known as the sequential Monte Carlo (MC) method are the more popular of the two approaches since they are able to deal with both non-Gaussian state and non-linear motion models [4]. Based on the Bayesian formulation, the PF employs a two-step process to track the target over time. The prediction step first recursively estimates the posterior distribution $p(x_t|z_{1:t-1})$ of the target state (x_t) using the motion model $p(x_t|x_{t-1})$ and the available observations $z_{1:t-1}$ as [16]:

$$p(x_t|z_{1:t-1}) = \int p(x_t|x_{t-1}) p(x_{t-1}|z_{1:t-1}) dx_{t-1}. \tag{2.9}$$

Then, using the observation at time t (z_t), the second step updates the target state as:

$$p(x_t|z_{1:t}) = \frac{p(z_t|x_t)p(x_t|z_{1:t-1})}{p(z_t|z_{1:t-1})}, \tag{2.10}$$

where $p(z_t|x_t)$ is the observation likelihood. The posterior $p(x_t|z_{1:t})$ in a PF, is approximated by a finite set of N particles $[x_{t,i}, p(z_t|x_{t,i})]_{i=1}^{N}$, where each particle state $x_{t,i}$ is weighted by its likelihood $p(z_t|x_{t,i})$. However, a common problem with PF is degeneracy, where after a few iterations, all but one particle will have negligible weights. To avoid this problem, the final resampling step eliminates particles with smaller weights and replicates particles with higher weights [4]. Increasing (decreasing) the number of particles can help in increasing (decreasing) the performance of PF based tracking methods, however a larger number of particles also implies a higher computational cost.

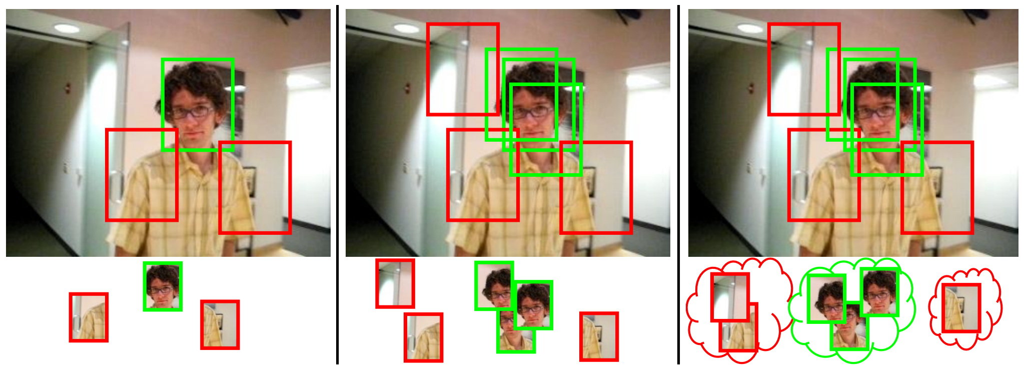


Figure 2.2: Representation of foreground (positive) and background (negative) image patches used by tracking-by-detection methods to train/or update the classifier. (a) Single foreground and multiple background patches, (b) multiple foreground and background patches, (c) bags of foreground and background patches where each bag contains several image patches. Cropped image is taken from the *David* sequence of the D1 dataset (Appendix A). This figure is an adaptation of Figure 1 from [6].

2.2.3 Generative and discriminative tracking methods

Generative and discriminative tracking methods are two separate classes of online learning based methods [156] that adapt to the target's appearance changes by updating the target model over time. Generative tracking methods typically work by learning a target model and then use the model to find the best matching image region [51, 142, 143]. Discriminative tracking methods commonly known as tracking-by-detection methods employ classifiers that discriminate the target from its surrounding background and search for the target within a specified region in the next frame [5, 6, 57, 58, 62, 84, 166, 166].

Tracking-by-detection methods formulate the tracking problem as repeated detections over time, where traditional approaches train a binary classifier to discriminate the target from the background. Image patches extracted near the target (foreground) and some farther away (background) are used to train the classifier (see Figure 2.2) and are assigned labels (1 and 0). The tracker searches for the target in each new frame within a local region around the previous target location and the patch with the maximum classification score is selected as the new target location. This patch is further used to update the appearance model and the remaining patches are discarded. However, the appearance model over time can degrade if it is updated with a suboptimal positive patch i.e. caused by an erroneous target estimation [6]. This can be handled by using the multiple instance learning framework [137], where the image patches are presented as bags (sets) of positive (foreground) and negative (background) samples (see Fig. 2.2) and each individual bag is assigned a binary label [6]. Using bags of patches in comparison to a single patch allows the method to train and update the classifier with less ambiguity [137]. Besides the traditional binary classifiers, other tracking-by-detection methods may employ classifiers such as those based on the structured output Support Vector Machine (SVM) [57, 132] or the correlation filters [10, 36, 62, 84]. Instead of assigning binary labels to the patches, the structured output SVM classifier (based on the *structured learning* approach) models the dependencies between all the available patches [19]. Correlation filters enhance the discriminative ability of the classifier by employing multiple negative samples [84]. However, these methods make use of signal processing techniques instead of the costly convolution operations used by traditional tracking-by-detection methods to keep the computational cost within bounds [130].

To further improve tracker stability and robustness, multiple weak classifiers can be combined into a unified classifier, commonly known as the classifier ensemble framework [5, 8, 132, 161]. Employing AdaBoost for combination, the strong classifier distinguishes between pixels belonging to the foreground and the background and generates a confidence map [5]. The target position in the next frame is then estimated by finding the peak of the confidence map using MS [5]. Moreover, weak classifiers can be combined either using performance-based weighted approach where the uncertainty of the classifier is used as its weight [8], or using an adaptive random subspace-based approach that adaptively adjusts the classifier weights based on their performance [161].

Despite the advances, visual tracking is still a difficult task due to challenges such as partial or full occlusions present in the tracking environment and different variations of the target. Trackers that employ motion models [93, 143] or that represent the target as multiple part-based patches [42] can handle occlusions better as compared to trackers that search for the target within a particular region [102]. Variations in target can include rotation, abrupt motion and changes in illumination, scale or pose. A tracker that uses colour information [103] may handle target rotations and scale changes well, but may fail due to illumination changes and background clutter. Combining colour features either with orientation [91] or texture [20] features can help the tracker to tackle these challenges.

2.2.4 Deep network-based tracking methods

Traditional tracking approaches use hand-crafted features that are built for specific scenarios and are unable to handle all the challenges associated with the tracking environment and the target [164]. More recently deep networks, which typically use raw data (images) to generate generic appearance representations, have gained considerable attention due to their improved performance in computer vision applications such as image classification [30, 82] and visual tracking [48, 107]. Methods for visual tracking generally use convolutional neural networks (CNNs) and recurrent neural networks (RNNs), where the former extracts features over the spatial domain and the latter has the capability to capture dependencies between features over the temporal domain [44].

Higher (deep) layers of a neural network extract abstract features while the more detailed features are better captured by the lower (shallow) layers of the network [44, 87, 144]. Fusing features extracted from both deep and shallow layers, can allow the method to better handle occlusions and avoid drift (caused by distractors) during tracking [87]. To reduce the computational complexity, a switch mechanism can also be employed to filter out redundant or noisy feature maps and only combine the most relevant ones [144]. Moreover, tracking performance can be further improved by fusing feature maps from both the spatial and temporal domains i.e. from CNNs and RNNs, respectively [44]. Furthermore, multiple trackers using features extracted from different layers of the network can be fused into a single strong tracker by combining their decisions in a performance-based weighted fashion [107].

Lack of training data is one of the main obstacles in the use of deep networks for visual tracking. Learning techniques based on standard classification tasks are not be applicable for such training due to the variations across video sequences that may lead to classifying the same object as foreground (target) in one sequence and as background in another sequence [97]. Multi-Domain Network (MDNet) an online tracking method uses offline training to learn the shared representation of targets from multiple annotated video sequences [97]. The method treats each training sequence as a separate domain and iteratively trains each domain individually to learn generic feature representations for visual tracking. A similar method uses a regression-based approach and an offline training phase to determine the different variations of both the target appearance and motion [60]. Since training a neural network is a slow process, methods that employ offline training have the advantage of being computationally efficient in comparison to methods that use online training. However, most existing neural network-based tracking methods address the deficiency of training data either by training online (from the first frame of the sequence) [35, 78, 145, 164] or by adopting pre-trained models (such as ImageNet [112]) from other tasks [37, 144]. Although visual tracking methods based on deep networks have proven to

Table 2.1: Summary of the selected trackers. KEY - ST: Spare-features based Tracker; AFT: Adaptive Fragments-based Tracker; LOT: Locally Orderless Tracker; IVT: Incremental Visual Tracker; AMS: Adaptive Mean Shift tracker; FCT: Fast Compressive Tracker; L1T: L1 Tracker; LSST: Least Soft-threshold Squares Tracker; IPCA: Incremental Principal Component Analysis; HSI: Hue Saturation Intensity; RGB: Red Green Blue.

Tracker	**Method**	**Tracker output**	**Target state**	**Model**		**Update**
				Appearance	**Motion**	
IVT [110]	probabilistic	bounding box	bounding box	intensity-based sparse model	Gaussian	incremental update by IPCA
ST [143]						occlusion-aware incremental update by IPCA
L1T [93]			multiple bounding boxes			appearance model
LSST [142]						
LOT [104]			super pixels based patches	HSI-based feature vector		
AFT [42]			part-based bounding boxes	intensity-based histograms	uniform	
FCT [166]	discriminative		multiple bounding boxes	intensity and Haar features	circular window search	
AMS [102]	deterministic	bounding ellipse	bounding ellipse	RGB-based histogram	implicit from data	none

be better than traditional tracking methods, the power of neural networks in visual tracking can be further utilized by training them on large-scale visual tracking datasets.

2.2.5 Selected trackers

To test our framework, we consider eight different trackers that have varying tracking models with different target representations, appearance models and motion models. Six of these trackers [42, 93, 104, 110, 142, 143] are based on the probabilistic framework (PF or its variants), one [102] on the deterministic tracking framework (MS), while the eighth [166] employs the tracking-by-detection approach. Our fusion framework uses only the outputs of the selected trackers making the framework transparent to the kind of trackers used. The six probabilistic trackers allows the method to take advantage of the particle filtering framework. Moreover, using a tracking-by-detection method [166] allows the fusion framework to handle situations where other trackers might fail (drift) due to distractors. We use the implementations publicly made available by the authors. Table 2.1 presents a summary of the trackers.

The first tracker is the *Sparse features-based Tracker* (ST) [143] and is based on the PF framework. Primarily motivated by the popularity of compressive sensing in visual processing, the method uses concepts of sparse coding [167] to generate the target appearance model. Utilizing both the sparse (intensity) features and the principal component analysis (PCA) subspace representation, the target appearance model is updated online to handle partial occlusions and

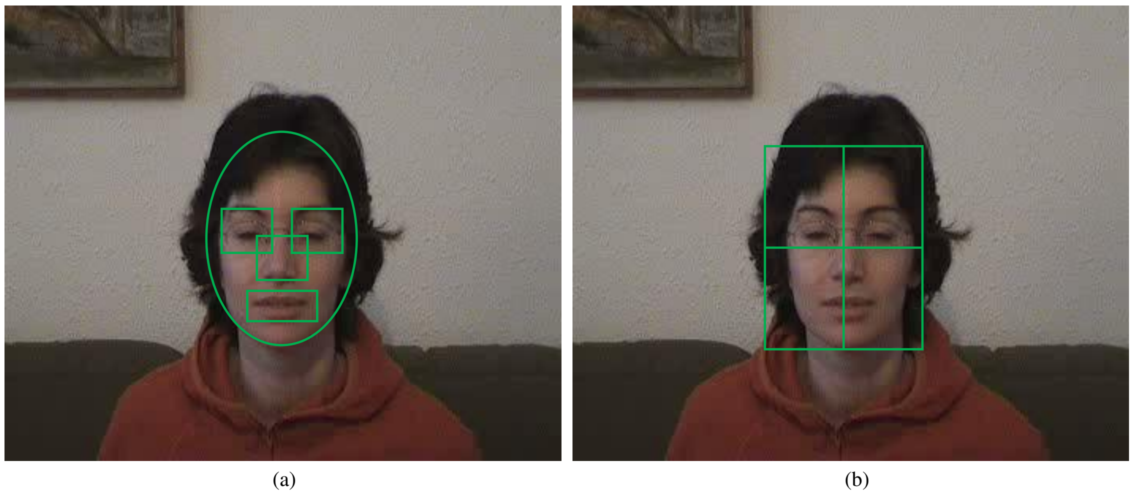


Figure 2.3: Target model based on multiple fragments, where each fragment is described by the same feature set. (a) An explicit part-based model, (b) a model-free approach. This figure is an adaptation of Figure 1(b) from [42])

illumination changes, while target motion is handled by employing the affine parameters. The final target state is selected using the Maximum a Posteriori (MAP) formulation i.e selection of the state from the set of hypothesized states with the highest likelihood.

The second tracker is the *Adaptive Fragments-based Tracker* (AFT) [42] which is based on the PF framework and divides the target model into multiple image patches (fragments). The method uses intensity histograms to model each image fragment, where these fragments are chosen arbitrarily i.e. without any specific model (Figure 2.3) and contribute to tracking based on their likelihood. The likelihood is computed by comparing the histograms of the image fragment with that of the respective template fragment using the Bhattacharya distance. Since each image fragment generates its own likelihood (confidence) value regarding the target's location, a joint likelihood of all the fragments is determined. Tracking is performed within a PF framework and the final target state is estimated as the likelihood-weighted average of the hypothesized states.

The third tracker is the *Locally Orderless Tracker* (LOT) [104] which employs a joint spatial and appearance model of the target. The method segments the target into super pixels [108], where each super pixel is modeled by its center position and the average HSV-colour values. Tracking is performed using a PF, where each particle state is weighted by its likelihood. The likelihood of each particle state is computed by calculating the Earth Movers Distance (EMD) [111] between the template patch and the observation patch. The final target state is determined as the likelihood-weighted average of the hypothesized states. The fourth tracker is the *Incremental Visual Tracker* (IVT) [110] that works by keeping an extended model of target appearances from the past within a sliding temporal window. Employing intensity values as the feature space, *eigen images* of the target are computed using incremental PCA. Tracking is performed using a PF framework and a Gaussian distribution around the previous target position determines the motion model. The likelihood of each hypothesized particle is determined by computing the distance between the target and the template model. The hypothesized state with the minimum score is selected as the final target state.

The fifth tracker is the *scale and orientation Adaptive MS Tracker* (AMS) [102], and extends the original MS tracking framework [32] by handling the scale and orientation changes of the target. The tracking algorithm utilizes RGB-colour histograms as features for the target model and performs matching by comparing the target and template histograms using the Bhattacharya distance. Target location in the next frame is obtained by estimating the mode of the function that maximizes the Bhattacharya distance. The method handles scale and orientation changes of the target by employing the zeroth-order and the second-order image moments, respectively.

The sixth tracker is the *Fast Compressive Tracker* (FCT) [166] with an appearance model based on features extracted from the compressed domain [165]. The method formulates tracking as a detection task and extracts multiple samples (image patches) of the foreground (positive) and background (negative) to build the appearance model. A multi-scale vector is created for each patch by convolving the patch with a set of rectangle filters at multiple scales. The method then constructs a high-dimensional feature matrix by concatenating the individual vectors. Employing the random projection technique [12], the method uses a spare random matrix to reduce (compress) the dimensionality of the feature matrix. Target position in the next frame is located by searching within a circular window based on the previous target location. Tracking is formulated as a binary classification task via a naive Bayesian classifier and the patch with the highest classifier score is selected as the final target state.

The seventh tracker is the *L1 Tracker* (L1T) [93], that models the appearance of the target by a sparse linear combination of target and trivial templates (set of unit vectors). Intensity values of windows sampled near the target and non-target intensity values (trivial templates) form the bases for a sparse representation, that is achieved by solving the L_1-regularized least squares problem. Assuming an affine motion model, tracking is performed within a PF framework by solving the L_1 minimization problem.

The eighth tracker is the *Least Soft-Threshold Squares Tracker* (LSST) [142], which is based on the PF framework. The appearance model is based on the sparse features that are obtained via the intensity values of the target. The LSS distance measures the dissimilarity between the target and the template model, that is determined by minimizing the error between the two models via linear regression. Using an affine motion model, the MAP formulation is used to determine the final target state.

2.3 Offline performance evaluation

Methods for offline performance evaluation of tracking algorithms estimate tracking quality over time using GT data [72, 98, 120, 125] by quantifying the accuracy or the robustness of a tracker. The accuracy can be measured as the distance between the estimated and ideal positions [125] or as the amount of overlap between the estimated and GT states [154]. Robustness measures the number of times a tracker fails during tracking [72], where the tracker is re-initialized after each failure. Furthermore, a weighted combination of the accuracy and robustness can be used to generate a combined tracking performance score [98]. The VOT[1] challenge provides a centralized benchmark for comparing different single target tracking algorithms. Moreover, surveys on video tracking covering the different SOA tracking algorithms either present their details and categories in general [90, 156, 158] or conduct experimental analysis over large benchmark datasets to assess their accuracy and robustness [72, 77, 125, 154, 155]. Although automatic GT annotation tools exist [11, 17], the annotations required can be extensive and expensive to produce for longer sequences and therefore cannot be applied for online performance evaluation.

2.4 Online performance evaluation

Methods for online performance evaluation [113, 114, 116, 149, C1] use information from the current and past time instants only and work without the requirement of any GT data. These methods can be used for performance evaluation of specific trackers such as PF [114, 116] or MS [117] based trackers or for generic tracking algorithms [27, 113, C1]. Moreover, online performance evaluation methods can be used in fusion frameworks for accuracy-based feature ranking [31], fine tuning tracking parameters [28], removing failing trackers [J1] or to correct low-performing trackers [16]. In this thesis we discuss methods of online performance evaluation as part of a framework to detect tracking failures (Chapter 3) and then to employ them within

[1]`http://www.votchallenge.net/`

Table 2.2: State-of-the-art methods employed for online performance evaluation in visual target tracking. Class (F/T) indicate whether the method has been applied to performance evaluation of features or tracking algorithms. KEY- F: Feature; T: Tracker.

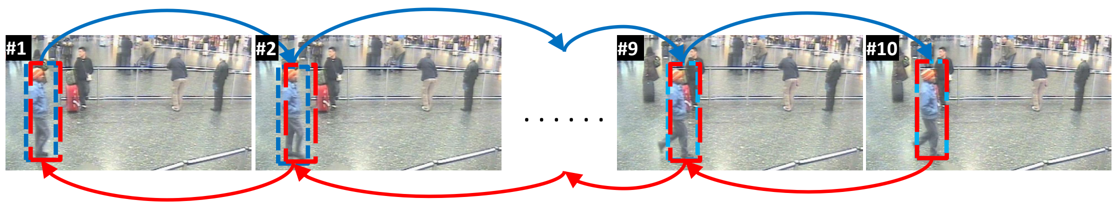
Ref.	**Category**	**Evaluation method**	**Features**	**Class**	
				F	**T**
[38, 123]	Trajectory	Temporal comparison	velocity		✓
[49]		Tracker motion + pair-wise correlation	position and size		✓
[149]		Time reversibility	state distribution		✓
[69, 116, J1]			position and state		✓
[14]			position	✓	✓
[7, 91]	Feature	Spatial uncertainty	colour and state distribution	✓	
[16, 114, 116]			colour and intensity		✓
[74, 160]		Observation likelihood	multiple		✓
[43, 88, 127, 133]		Democratic integration	multiple	✓	✓
[41]		Spatial difference along edges	colour and motion	✓	
[31]		Background discriminative power	colour and intensity	✓	
[113, C1]		Matching descriptors	colour and position		✓
[69, 105]	Hybrid	Time reversibility + temporal comparison	colour and position		✓
[116]		Time reversibility + spatial uncertainty	colour, position and size		✓
[18, 25–28, 126, 150]		Temporal comparison	multiple		✓

a tracker-level fusion framework (Chapter 4). This evaluation can either be performed using trajectories (tracker output), features (internal or external to the tracker) or hybrid (combination of both). Table 2.2 gives a summary of the different performance evaluation methods that employ trajectories, features or both in a visual tracking framework.

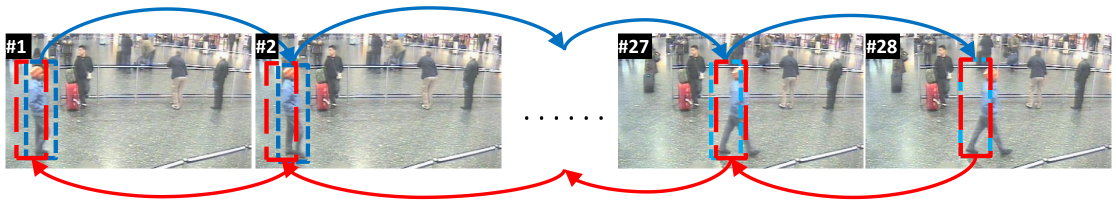
2.4.1 Trajectory-based performance evaluators

Trajectory-based performance evaluators consider the physical properties of the target state such as velocity [26, 38, 123], area (number of pixels within the bounding box) [26, 28, 126], motion smoothness [115] or direction [18, 26] or can use additional trajectories such as spatial agreement between the output of a set of trackers [49] or compare results of the tracker with the same tracker running in reverse direction [14, 69, 115, 149, J1].

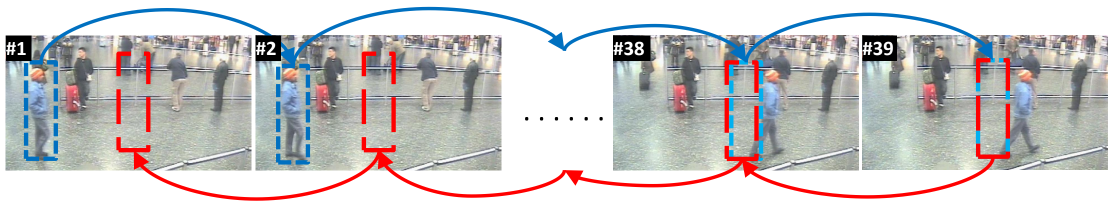
Physical properties of the target state such as the target displacement, velocity, direction, shape or area [26] can be used to evaluate tracking performance. Tracking performance is generally determined by measuring the temporal changes of these physical properties and comparing them with different thresholds, where values higher than the threshold indicate a possible tracking failure [26, 38, 123]. For example, the target displacement between two consecutive frames can either be compared to a fixed threshold [26] or to an adaptive threshold (i.e. the average target displacement within a sliding temporal window) [123]. Similarly, tracking performance can also be estimated by quantifying either the ratio of the target area or the direction of a moving target between two consecutive frames and comparing them to pre-defined thresholds [18, 26]. How-



(a)



(b)



(c)

Figure 2.4: The time reversed evaluation approach [149]. For each frame where the tracker runs in the forward direction, the same tracker is run in the backward direction up until the reference (first) frame. - - -: Forward tracker; – – –: Reverse tracker. The figure shows the reverse tracker initialised at frame (a) 10, (b) 28, and (c) 39.

ever, majority of these methods use fixed pre-defined thresholds [26, 115] that implicitly implies strong assumptions on the moving direction, area and velocity of the target [18, 26, 123].

Within a tracker-level fusion framework, tracking performance is determined by combining individual tracker quality and agreement between the trackers [49]. Individual tracker quality is measured as the amount of overlap of the tracker output between two consecutive frames. However, overlap of tracker outputs between consecutive frames can produce incorrect results due to *tracker drift* (a wrong estimation from the tracker always returns a high performance score even when the tracker has lost the target). Agreement between tracking outputs is computed as the spatial pair-wise correlation score, however this can lead to misleading results in situations where majority of the trackers perform badly.

Based on the time-reversible property of Markov chains [149], tracking performance is evaluated by using another tracker (reverse tracker) that runs in the reverse direction until a reference frame in which the tracker under evaluation (forward) is assumed to be correct. The reference frame is assumed to be the first frame, where the forward tracker is initialised by the GT. The result from forward tracker at time t is used as the initialization for the reverse tracker, which is then used to compute the tracking result of the reverse tracker at the reference frame. Taking the assumption that the forward and reverse distributions show similar statistical properties, the results from the forward and reverse trackers at the reference frame are compared using the Mahalanobis distance in a PF framework [149]. This method has been employed in various scenarios such as using full length trajectories [69], shortened trajectories [116, J1], template matching on a frame-by-frame basis [83] or for validating target scale changes in an MS tracking framework [140]. However, the method has two main drawbacks; first it is prone to tracker drift where errors from forward tracker are accumulated by the reverse tracker and second the computational complexity of the approach limits its use to real-time systems. An example of the method is presented in Figure 2.4, showing three different time instants where the reverse tracker is initialised. Figure 2.4 (a) and (b) present the scenario where the reverse tracker is initialised with a correct estimation of the forward tracker and is therefore able to accurately estimate the target's position in the first frame. Whereas Figure 2.4 (c) indicates a scenario where the reverse tracker is being initialised by an incorrect estimation and accumulates the errors generated by the forward tracker.

2.4.2 Feature-based performance evaluators

Feature-based performance evaluators can make use of either *tracker-independent* features [113, C1] or the internal properties of the trackers such as observation likelihood [74, 134], spatial uncertainty of PFs [91,114,116], uncertainty of target distribution in a MS tracking framework [117] or spatial differences along edges [41].

Tracker-independent features use the tracker output and extract image features based on the tracker state. Using a five-dimensional descriptor based on the position information (two dimensions) and colour values (three dimensions), the target appearance structure is measured via the covariance feature [113]. A temporal comparison of these covariance features is then employed to measure the tracking performance. A similar approach measures tracker performance by target-background discrimination [C1]. The method determines distributions of both the target and a larger region around the tracker state using colour-based distribution fields [146]. It then measures tracking performance by comparing the two distributions using the L_1 distance [C1]. *Tracker-independent* methods have the advantage that they can be used with any kind of tracker(s), since they only require the output of the tracker(s), but matching descriptors over time can generate false positives and false negatives due to clutter or similarity between descriptors.

By employing the features that are part of the trackers, properties such as the observation likelihood [74, 160] are used to select the best tracker with the highest likelihood, or the spatial uncertainty (spread of the particles) of the PFs can be used to either determine the performance of a tracker [114, 116, 117], determine the best trackers [7, 16] or assign performance weights to the features [91, 151]. The spatial uncertainty of a PF is determined by computing the covariance matrix $\mathbf{C}_t$ of its particle states $[x_{t,i}]_{i=1}^{N}$ weighted by their respective likelihoods $p(z|x_{t,i})$. The 2x2 normalized covariance matrix $\mathbf{C}_t$ for a two dimensional state $x_t = (u_t, v_t)$ is given as [91]:

$$\mathbf{C}_t = \begin{bmatrix} \frac{\sum_{i=1}^{N} p(z|x_{t,i})(u_{t,i}-\hat{u})^2}{\sum_{i=1}^{N} p(z_t|x_{t,i})} & \frac{\sum_{i=1}^{N} p(z|x_{t,i})(u_{t,i}-\hat{u})(v_{t,i}-\hat{v})}{\sum_{i=1}^{N} p(z|x_{t,i})} \\ \frac{\sum_{i=1}^{N} p(z|x_{t,i})(u_{t,i}-\hat{u})(v_{t,i}-\hat{v})}{\sum_{i=1}^{N} p(z|x_{t,i})} & \frac{\sum_{i=1}^{N} p(z|x_{t,i})(v_{t,i}-\hat{v})^2}{\sum_{i=1}^{N} p(z|x_{t,i})} \end{bmatrix}, \tag{2.11}$$

where $\hat{u}$ and $\hat{v}$ are the average state vectors of the particles. The spatial uncertainty is then computed as $S_t = \sqrt[D]{det\,(\mathbf{C}_t)}$, where det(.) is the determinant of $\mathbf{C}_t$ and D is the dimensionality of the state space. The spatial uncertainty has been employed in different situations, such as to determine the performance of a single tracker [114, 116]. These methods normalise the uncertainty values with the tracker state size and then employ sliding temporal windows with varying lengths to evaluate tracker performance by analysing the temporal changes within the uncertainty signal. Within a multi-tracker framework, spatial uncertainty of each tracker can either help in switching to the best tracker [7] or computing a weighted sum of the tracking outputs [16]. When combining multiple features within a single tracking framework the spatial uncertainty helps in computing the performance of individual features, where the performance score is computed as the reciprocal of the uncertainty [91, 151] and the likelihood of each feature is determined as the distance from the target model histograms. However, methods based on the observation likelihood or the uncertainty are limited to PF-based trackers. Furthermore, distractors (i.e. objects with features similar to those of the target) may produce high likelihood values thus generating misleading measurements. An example of using the spatial uncertainty for performance evaluation is shown in Figure 2.5.

Tracking performance can also be evaluated using features that are not the internal properties of the tracker itself, but are internally embedded within the tracking framework. This can either be achieved by using the target-background discrimination approach [31, 39, 41, 55, 56]

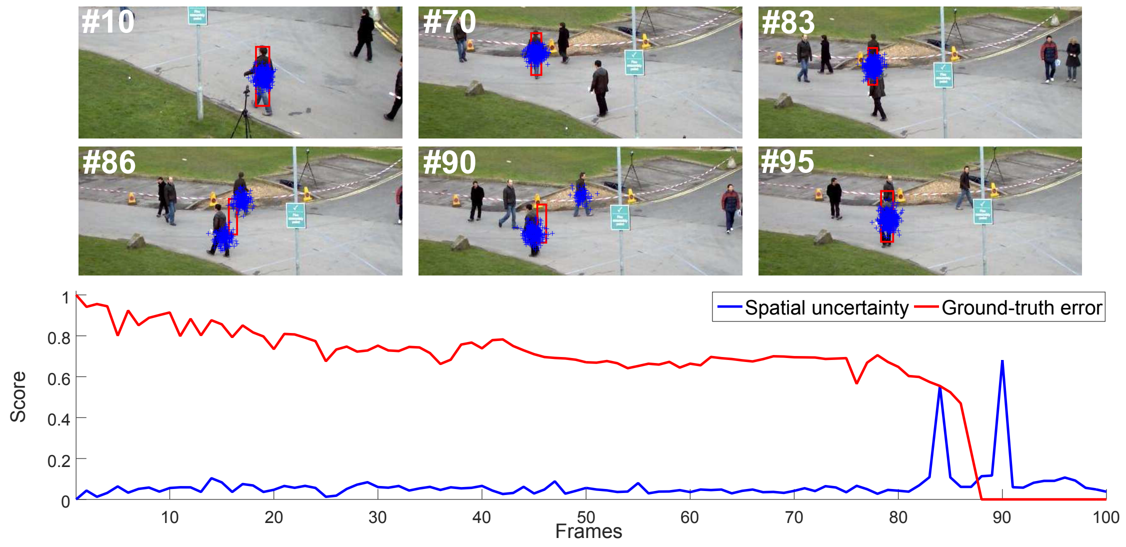


Figure 2.5: Performance evaluation of the tracker LOT measured as its spatial uncertainty [91]. Sample tracking results and spatial location of particles are represented as red bounding box and blue crosses for each frame, respectively. Ground-truth error is computed as the overlap score described in Section 3.4.2.

or using distance based measures [23, 88, 127, 133]. For the target-background discrimination approach, a two-class variance ratio method ranks features based on their ability to distinguish the target from the background [31], the spatial differences of colour and motion features along the object boundary determine tracking performance of each individual feature [41], or tracking performance is measured by combining colour and contour features within a KF [55] or PF [56] framework. Distance-based measures can be used to compute the minimum squared distance between the reference and target histograms as the feature weight [23] or use the *democratic integration* [133] in a PF framework [43, 127] or an MS framework [88]. *Democratic integration* provides a way to both evaluate performance and combine features, where the individual feature result is compared to the agreed (fused) result allowing a higher weight to features that remain consistent with the agreed result in the recent past. This is performed via a feedback loop, where the fused result is fed back to each individual feature and performance weight of each feature is determined by quantifying the degree of agreement between the results from the individual feature and the fused output. This also allows the features to adaptively adjust their performance. However, a drawback of the feedback loop is the tendency of adjusting to an incorrect estimation, causing the features to adjust towards the wrong result.

2.4.3 Hybrid performance evaluators

Hybrid measures combine both trajectory and feature based performance evaluators to evaluate tracking performance. For example, both the likelihood of the PF and the smoothness of target velocity can be used to determine when the performance of the tracker deteriorates [38]. Moreover, the time-reversed evaluation approach can be jointly employed with spatial uncertainty [116] or be combined with temporal difference of colour histograms in an equally weighted approach [69, 105]. The temporal difference of colour histograms are also combined with shape consistency and motion and direction smoothness to evaluate tracking performance [18]. Furthermore, different sets of feature and trajectory-based performance evaluators can be combined for determining either a global tracking performance score [26, 126, 150] or the optimal tracking parameters [25, 28]. These performance evaluators can be based on measuring either the temporal similarity of the target's appearance or the temporal consistency of the target's shape, velocity and area. Then a global tracking performance score can be determined by combining the evaluators in a weighted average approach [26], in a weighted sum approach [150] or by using a naive Bayes classifier [126]. The optimal parameters required for robust tracking can be determined by evaluating tracking performance using a set of features that are based on colour intensity differences between the target and background, target area, density of objects (i.e. sum of areas of all objects within the scene) and the area overlap between objects [25, 28]. However, these approaches are either dependent on pre-defined thresholds [26, 28, 126, 150] or require an offline training phase to learn the optimal tracking parameters [25, 28].

2.4.4 Tracking failure detection

Failure detection refers to determining time instants at which the tracker fails by employing one or more of the performance evaluation techniques discussed in Section 2.4.1-2.4.3. Employing the spatial uncertainty of PFs [116], tracker failure is detected by analysing temporal changes of the uncertainty signal over two separate sliding time windows that handle short-term and long-term changes, respectively. The method employs a finite state machine to switch between tracking failure and recovery modes, and detects failure using empirical thresholds. A similar approach employs a change-detection approach to determine time instants of tracking failure and models the changes of the uncertainty signal via a mixture of Gamma distributions [114]. However, approaches based on the spatial uncertainty are tracker-specific. Tracker-independent methods use only the tracker state to either compute feature descriptors [113, C1] or to generate another tracker trajectory that runs in the time-reversed direction [149]. Tracking failure is detected by employing thresholds over the tracking performance score generated by comparing either the covariance features over consecutive frames [113], the target-background feature descriptors at each time instant [C1] or the forward-reverse tracking results at a specific frame [149]. However, using experimentally derived thresholds [113, 116, 149, C1] may limit the methods to specific trackers or visual data.

2.5 Fusion for visual tracking

Fusion enables the combining of data or information from multiple sources such that the resulting data is more accurate and useful in comparison to data obtained from any single source [71]. The field of visual tracking presents multiple challenges where each individual tracker is developed to handle particular sub-challenges [125]. Given that a visual tracker can fail under certain challenges, the primary objective of fusion in visual tracking is to improve the overall tracking performance.

Fusion in visual tracking can be applied at feature or tracker-level and the main objective of both approaches is to compensate the failure of a single feature or tracker by using multiple complementary features or trackers, respectively. Methods of feature-level fusion achieve this objective by combining multiple features in a single tracking method [43, 62, 127, 151]. The combination is performed internally to generate a robust target appearance model allowing the tracker to handle multiple visual challenges [91]. Tracker-level fusion combines the decisions (outputs) of different trackers to achieve a similar task i.e. handle multiple visual challenges [16, 49, 118, J1]. The problem is formulated by estimating the relationships between the tracker outputs where each single-feature tracker is modelled as a black box.

Fusion can be performed using simple combination rules such as the *product* [23, 76] or the *sum rule* [16, 49, 151]. The *product rule* generally is effective for combining features that are independent of each other (e.g. colour, edge and texture features [23]). Within a particle filtering framework the product rule generates an overall likelihood function of the particle filter by combining the likelihoods of the individual feature, where the contribution of the likelihood can either be non-weighted [76] or weighted based on the individual feature performance [23]. The *sum rule* combines outputs or decisions generated by different sources by assigning them equal weights [9,13,79,J1] or weights according to their individual performances [16,49,91,127,151]. Moreover the *sum rule* in comparison to the *product rule* is less sensitive to clutter and occlusions [151]. Other common techniques used for fusion include the *interaction and sampling*-based [39, 74, 96] or *correction*-based approaches [118, 123]. *Interaction and sampling*-based approaches achieve combination of multiple features [39, 96, 106] or trackers [74, 118, 124] that share information (particles, observation likelihoods or state estimates) between each other to improve the overall tracking performance. *Correction*-based approaches employ multiple trackers that work together to improve (correct) each others' state estimations hence improving the overall tracking performance [118, 124]. In the following subsections, we present and discuss details of feature and tracker-level fusion methods.

2.5.1 Feature-level fusion

Fusion at feature level is generally carried out under a single tracker framework [91], where multiple visual features are combined together to improve target representation as compared to using a single feature. Different sources with complementary performance working together generate better results, where these sources could either be the features such as colour, texture, shape, motion, edges or points obtained from a single sensor or features obtained from different physical sensors. A comprehensive survey on feature-level fusion for object tracking covers the fusion aspect based on the feature acquisition sources i.e. from a single sensor (e.g. vision or thermal cameras) and from multiple sensors (e.g. vision, thermal, laser, radio, stereo) [141]. In this section we only consider fusion of features extracted from a single source (vision camera). The features can be combined either without determining their performance (equal weighting) [13, 106] or by determining their performance weights [31, 68, 91, 127] using the performance evaluators as described in Section 2.4. Another approach to feature-level fusion allows the features to work together by exchanging information with each other within a cooperative framework [39, 96]. A summary of the state-of-the-art methods for feature level fusion is presented in Table 2.3.

One of the earliest approaches to feature level fusion combines colour histograms and intensity gradients for head tracking by giving the features equal weights [13]. Being statistically independent, the two features complement each other and provide a matching score, where the final result is obtained by taking a direct sum of matching scores. A similar approach uses texture descriptors and colour histograms with equal weights in a MS tracking framework [20]. However, methods where all features are given equal weight can generate incorrect results in case

Table 2.3: State-of-the-art methods used for feature level fusion in visual target tracking. Key - FS: Full Search; PF: Particle Filter; KF: Kalman Filter; MS: Mean Shift; PHD: Probability Hypothesis Density filter; IPCA: Incremental Principle Component Analysis [110]; SIFT: Scale-Invariant Feature Transform.

Ref	**Tracking**	**Target feature(s)**	**Performance Evaluation**	**Fusion method**
[13]	FS	colour, intensity gradients	none	Linear sum
[96]	KF, PF	colour, contour	none	Interaction between filters
[20]	MS	colour, texture	none	Individual use
[88]	MS	colour, motion	Democratic integration	Democratic integration
[23]	PF	colour, edge, texture	Feature likelihood	Weighted product
[162]	PF	colour, IPCA	Background discriminative power	Weighted exponential function
[31]	MS	colour, intensity	Background discriminative power	Rank based feature selection
[133]	FS	colour, motion, contrast, shape, position	Democratic integration	Weighted sum
[127]	PF	colour, intensity	Democratic integration	
[151]	PHD	colour, SIFT	Spatial uncertainty	
[91]	PF	colour, orientation	Spatial uncertainty	
[68]	PF	colour, intensity, texture	Background discriminative power	
[106]	PF	colour, motion, sound	none	Partitioned sampling
[152]	PF	colour, shape	none	
[39]	PF	colour, edge, contours	Background discriminative power	Dynamic partitioned sampling

when one or more features become unreliable.

Assigning performance-based scores to the features allows the method to use those features that are more reliable. *Democratic integration* has been widely used for both weighting and/or combining different features within a tracking framework [88, 127, 133]. An advantage of *democratic integration* is its feedback loop, which allows features to improve their performance at each time step based on the comparison between the result from each individual feature and the combined result. The motion, colour, position, shape and contrast features are weighted using the *democratic integration* approach and combined by a weighted sum of their probability densities in a face tracking framework [133]. Using the same approach for performance evaluation and fusion a self-organising method combines colour and intensity features for visual tracking [127], while the approach of *democratic integration* can also be used to both weight and fuse colour and motion cues within an MS tracking framework [88]. Features can also be fused by performance weighting using either their likelihood [23] or their spatial uncertainty [91, 151]. Employing the weighted product of likelihoods as the fusion approach, colour, texture and edge features are combined under a Bayesian framework using feature likelihood as the performance score [23]. Similarly, the spatial uncertainty has been employed in a PF tracking framework to weight colour and orientation features [91] and in a Probability Hypothesis Density (PHD) filtering framework to weight colour and SIFT features [151], while fusion is carried out using the weighted sum rule. However, using the likelihood or spatial uncertainty as feature weights can produce misleading results if the tracker follows a distractor.

Another approach to feature-level fusion works by sharing information between the features [96, 106, 152] . These methods integrate the features within the prediction step and the interaction between features occurs during the tracking process. With four different filters (one KF and three PF), colour and contour features are used together to track the target by separating it from the background [96]. Each filter uses a specific feature to separately estimate the target state, while also being dependent on the posterior of the feature returned by the previous filter. Employing a partitioned sampling approach, colour features are combined with sound for teleconferencing and with motion for surveillance [106]. The sampling is a two layer process where the likelihood of the sound or motion feature provides an approximate location of the target on the first layer, while the colour feature improves the result on the second layer. Another approach uses the colour and shape features where each receive priors from the other one [152]. Then each feature draws its samples based on its own likelihood and the priors received from the other feature, hence increasing the likelihood of each feature. However these methods present the problem of having a fixed order of the partitions, which can be solved by the Dynamic Partitioned Sampling (DPS) approach [39]. DPS allows the order of the partitions to be dynamically changed based on the performance weight of each feature. The method uses colour, edge and contours as the features and weights the performance of each feature based on its ability to discriminate the object from the background. The discriminative model (ability of the features to discriminate the object from the background) can also be used to combine confidence maps of each individual features [68]. The confidence maps of the colour, intensity and texture features are fused into a single confidence map using the weighted sum rule [68]. The discriminative model can also be used to rank and select the best features at run time [31].

2.5.2 Tracker-level fusion

An individual tracker can be designed to be robust against occlusions but might be unable to deal with orientation changes of the target. Similarly another tracker could be robust against illumination and scale changes, but fail to handle background clutter. Therefore, the concept of combining multiple trackers is to handle multiple visual challenges within a single framework. Trackers can be combined under a cascade (sequential) [89,118,124] or in a parallel [49,59,76,J1] framework. Trackers that are dependent on the output from another tracker or where a tracker is integrated into another tracker [89] are categorized as cascade-based methods. Parallel frameworks for tracker-level fusion allow the trackers to run independently and combine the outputs from each tracker. Finally, tracker interaction and sampling based approaches [74] work in a cooperative framework where the best tracker is chosen based on its performance. Table 2.4 gives a summary of the different tracker-level fusion methods used for combining multiple trackers under a single framework.

Sequential execution of multiple tracking algorithms requires that trackers are used in a specific order [118, 123, 124]. A tracking-by-detection approach, handles the problem of tracker drift by combining a template-based tracker, an optical-flow based MS tracker and on-line random forest tracker in a fall-back cascade [118]. Using a loop-back cascade another approach uses two trackers (based on region and shape) and two detectors (head and motion) for people tracking [124]. The four modules interact with each other having access to the output from other modules, as well as to long-term tracking history. This approach however, limits the methods to a specific trackers and order of execution which can be crucial if changed.

Parallel frameworks for tracker-level fusion generally consider the trackers as black boxes, using only the outputs from each individual tracker and are not dependent upon the type of trackers employed in the framework. In a parallel framework, tracking algorithms can be combined using their Probability Density Function (PDF) without evaluating their performance, where the fused output is estimated as the product of the PDFs [76]. Performance evaluation of the trackers within a parallel fusion framework makes it possible to either weight trackers [16,49,59], correct weak trackers [16, 123] or remove failing trackers [J1]. Tracking outputs can then be combined via the sum rule using either the trackers' performance weights [16,49] or equal weights [J1]. A different approach to parallel tracker fusion can either make use of an additional detector within the framework [139] or generate a shared pixel based representation using the individual tracking outputs [59]. Employing an additional detector, the method estimates tracking performance and combines the strengths of three different trackers within a Hidden Markov Model (HMM) framework [139]. Tracking performance is measured by two different techniques; internal (dependent) and external (independent) to the trackers used. Internal tracking performance is generated by the trackers individually while external performance is measured by computing the distance between the target and reference model histograms. Tracking outputs can also be fused by generating individual tracking support sets that are then combined into a shared-pixel based representation [59]. Each support set is defined by the coordinates and corresponding likelihoods of the pixels contained within the output of each tracker. The combination of the support sets is based on the

Table 2.4: State-of-the-art for tracker-level fusion. Key - S: Spatial; T: Temporal; C: Cascade; P: Parallel; TM: Template Matching; ORF: Online Random Forest Tracker; MS: Mean Shift; RT: Region Tracker; AST: Active Shape Tracker; PF: Particle Filter; EBT: Edge based Tracker; FoT: Flock of Trackers [138]; ASMS: Scale Adaptive MST [140]; KCF: Kernelized Correlation Filter [62]; FT: Fragments-based Tracker [2]; OBT: Online Boosting Tracker; SOBT: Semi-supervised OBT; BSOBT: Beyond SOBT; SURFT: SURF-based Tracker; CHPF: Colour Histogram PF; HPF: HOG-based PF; BT: Background-based Tracker; OFT: Optical flow-based Tracker; STR: STRUCK Tracker [57, 58]; ASST: Adaptive Structural Sparse-based Tracker; HOGHT: Hough Transform based tracker; BTT: Blending based Template tracker; MIL: Multiple Instance Learning Tracker [6]; SB: SemiBoost; IVT: Incremental Visual Tracker [51]; IVTE: IVT using Edge; ST: Sparse-based Tracker [143]; AFT: Adaptive FT [42]; LOT: Locally Orderless Tracker [104]; FCT: Fast Compressive Tracker [166]; AMS: Adaptive MST [102]; L1T: L1-based Tracker [93]; LSST: Least Soft Squares threshold Tracker [142]; PDF: Probability Density Function; HMM: Hidden Markov Model; GLAD: Generative model of Labels, Abilities and Difficulties [148]; DN: Deep Network.

Ref	**Type**	**Num. of trackers**	**Tracker(s)**	**Performance evaluation**	**Tracker correlation**		**Fusion method**	**Fused trackers**
					S	**T**		
[118]	C	3	TM, ORF, MS	none			Correction	All
[124]		2	RT, AST	none			Interaction	
[54, 89, 168]		2	PF, MS	none			Kernel-Bayesian	
[76]	P	2	PF	none			Product of PDFs	
[123]		3	RT, EBT	none			Correction	
[139]		3	FoT, ASMS, KCF	Histogram distance			HMM	
[169, 170]		6	FT, OBT, SOBT, BSOBT, MIL, SURFT	Distance to fusion	✓		GLAD	
[49]		13	CHPF, HPF, MS, OBT, SOBT, BSOBT, MIL, BT, OFT, IVT, STR, ASST, L1T	Tracker motion	✓		Weighted sum	
[16]		2	PF	Spatial uncertainty				
[59]		3	HT, MS, BTT	Distance to fusion				
[107]		10	DN-based correlation filters	Adaptive Hedge [29]				
[79]		4	MIL, SB, IVT, IVTE	Spatial disagreement	✓		Democratic integration	Selected
[74]		8	PF	Likelihood			Interaction and sampling	
[160]		4						
[159]		4						
[J1]		8	ST, AFT, LOT, IVT, FCT, AMS, L1T, LSST	Time reversibility	✓	✓	Average	

weighted sum rule, where the weight is determined by overlap of a tracker's support set with the fused support set. Parallel execution order allows to add, remove or re-order trackers if required, hence is not limited by specific order or trackers. A hybrid approach that combines both parallel and sequential execution, evaluates the performance of each tracker separately and then learns which approach is useful for a given scenario [129]. Using the confidence scores of the features within each tracker, the tracker with the highest score is selected in parallel execution whereas in sequential order the first tracker with an expected error value below a fixed threshold is selected.

Besides combining the strengths of trackers in sequential or parallel orders, the Kernel-Bayesian framework works by integrating a tracker into the framework of another tracker [24, 54, 89, 168]. The framework makes it possible to utilize the accuracy of the PF as well as the low complexity of the MS tracker [89] by employing the MS tracker in the prediction stage for improving the prediction for the PF [168]. This helps in reducing the computational complexity of the conventional PF and improves reliability of the MS tracker. However, similar to the sequential execution order, the Kernel-Bayesian framework is also limited to a specific type and order of trackers.

Interaction and sampling based approaches are based on a probabilistic framework, where multiple trackers interact with each other by sharing their information and are sampled to select an individual tracker according to the situation. Sampling the trackers based on their appearance and motion models via the Markov Chain Monte Carlo (MCMC) method, the trackers further share information with each other while the appropriate tracker is selected based on its likelihood score [74]. A similar approach employs the motion and the observation likelihood models for tracker interaction and sampling, where the tracker with the highest likelihood is selected [159, 160]. Using the Annealed PF (APF) and the Gaussian Process Dynamic Model (GPDM) over a low-dimensional tracker, target motion models are utilised to sample both trackers [85]. The two trackers work by communicating and exchanging particles with each other. The GPDM based tracker is trained to learn the motion models and therefore the sampler automatically switches to the GPDM tracker for trained sequences, while it assigns a higher weight to the APF for sequences that represent un-trained motions. The process of tracker interaction makes it possible to prevent failing trackers from drifting [160], but sampling based approaches can become computationally complex and expensive as the size of the state or number of trackers increases. Other approaches to tracker-level fusion include learning and classifier-based methods [79,169], where labels (foreground/background) are assigned to image patches, while tracker performance is measured either as the distance between the fused output and the output of each tracker [169, 170] or as disagreement from other trackers [79].

2.6 Discussion

This chapter presented a review of the state-of-the-art in visual tracking, tracking performance evaluation and fusion for visual tracking. The section for visual tracking covers different types of tracking methods and details of the visual trackers employed for experimental evaluations of our framework. Performance evaluation is divided into offline and online performance evaluation methods, where a detailed discussion on trajectory, feature and hybrid-based online performance evaluators, including tracking failure detection methods, is covered. Finally, fusion for visual tracking covers both feature-level and tracker-level fusion methods.

Changes in the visual tracking environment as well as that of the target can reduce tracking performance or cause tracker failure. However, fusion in visual tracking enables the combining of multiple tracking algorithms in order to improve the overall tracking performance. Fusion at *feature-level* uses multiple features to generate a target appearance model that is robust to the challenges present in visual data [20,91,151]. Nonetheless, with increasing number and dimensionality of features, the complexity of such methods increases. *Tracker-level* fusion methods provide a less complex approach where outputs (decisions) from multiple trackers can be combined in a *cascade* [118,124] or in a *parallel* [16,49,J1] fashion. *Interaction and sampling* based approaches [74,85,159] provide another way of combining multiple trackers, where the trackers share information and are sampled to select an individual tracker according to the situation. *Cascade*-based approaches propose a dependence of trackers on each other where the output from one tracker is used as input for the next tracker [118,124] or a tracker is integrated within another tracker such as in the Kernel-Bayesian framework [54,89,168]. These methods imply the use of either a specific execution order or specific trackers and are limited by the pre-definition of the number of trackers. *Interaction and sampling* based methods [74, 85, 159] are based on the PF framework, where multiple trackers work by sharing information (particles) with each other. Although not limited to any specific execution order, these methods are restricted to the use of PFs. *Parallel*-based approaches [16,49,J1] provide the flexibility to include or remove different types of trackers when required and are not limited by the execution order since each tracker works independently. The outputs from each tracker are then combined to estimate the final target state.

Furthermore, evaluating tracking performance at run-time (online) can provide additional information to either assign performance-based weights during fusion [16, 49] or to select the best performing one from a set of trackers [74]. These performance evaluators can make use of the *features* [16, 113, 115, C1], the *trajectories* [49, 123, 149, J1], or a combination of both (*hybrid*) [28, 38, 116]. Performance evaluators based on *features* such as the spatial uncertainty [7,16,91] or the observation likelihood [74,159] make the evaluator dependent on specific trackers (PFs). *Trajectory*-based performance evaluators using the physical properties of the target such as direction [18], velocity [123], area or shape [26] can restrict the methods to specific visual data. The time-reversed evaluation [14, 69, 149] method evaluates performance by comparing outputs of the tracker (forward tracker) under evaluation with another tracker that runs in the time-reversed direction until the reference frame i.e. where the forward tracker is assumed to be correct. Albeit effective, this approach imposes a heavy computational cost as the first frame is selected to be the reference frame, where the forward tracker is initialised by GT.

Moreover, these performance evaluators can be employed to detect tracking failures [113, 116], for example by employing empirical thresholds either over temporal changes of the spatial uncertainty [114, 116] or temporal comparisons of covariance descriptors [113]. Despite promising results for failure detection, the methods can either be restricted to specific trackers [114, 116] or can be limited to specific data due to the use of empirically derived thresholds [113]. Furthermore, the variable range of values over different trackers and visual data also limits the use of these performance evaluators.

To address the above mentioned limitations of the SOA, we present a parallel framework for tracker-level fusion. Due to the challenges involved, trackers can fail over time causing them to drift away from the target and hence create different clusters (groups of trackers sharing similar spatial locations). Unlike the SOA methods that use all trackers for fusion [16, 49, 139] or only select the best one [74, 159], our framework employs a clustering approach that determines the clusters of trackers over time. The clustering approach makes use of short-term temporal data (from preceding frames) to determine the pairwise spatio-temporal relationships between trackers in order to determine the different clusters. Then using the time-reversed evaluation approach we identify and discard failing trackers and use only the ones following the correct target (*on-target*). In comparison to the time-reversed evaluation approach [149] that uses the first frame as the reference frame, we adaptively update the reference frame forward in time to limit the computational cost (Chapter 4). We combine the outputs from the *on-target* trackers using a performance-based weighted approach, where tracker performance is determined using a feature-based online performance evaluator (Chapter 3). The performance evaluator makes use of features that are determined using standard tracker outputs making the method tracker-independent [C1]. We first evaluate tracking accuracy over time by comparing distributions of the image region defined by the tracker state and a larger region around the state. Then, we temporally smooth the accuracy score using a forecasting approach which allows the accuracy score to remain flexible over different trackers and visual data as demonstrated by the experimental results in Section 3.4. Finally, we employ an experimentally trained threshold over the tracker accuracy score to detect tracking failures.

Chapter 3

Tracking failure detection via forecasting

3.1 Introduction

Time instants at which a tracker fails can be estimated by validating the tracking performance over time. Tracking performance at run time can be measured by using tracker-dependent features such as the observation likelihood [74] or the filter uncertainty [16, 116]. Features independent of the tracker make use of the tracking outputs to extract features such the covariance descriptors [113] from the image. These techniques can then be employed to detect time instants of tracking failure [113, 114, 116].

In this chapter, we propose a framework that first evaluates tracking performance by employing tracker-independent features. The method works on the state-background discrimination approach, where we use the tracker output to extract features of the state and the background region around the state. Then, we determine distributions of the state and the background using Distribution Fields (DF) [100, 122, 146], where DF represents a smoothed histogram of the image region composed of several layers (bins). We compare the state and background distributions to quantify the similarity between the two regions, thus generating a track performance score. Raw (noisy) values of the tracking performance score can have variable ranges for different sequences and trackers, thereby limiting the SOA methods to specific sequences or trackers [113, 116]. To address this problem and detect tracking failures, we model the score as a time series using the Auto Regressive Moving Average (ARMA) model [22] and forecast future values of the time series. The difference between the original and the forecast generates a forecast error signal, which

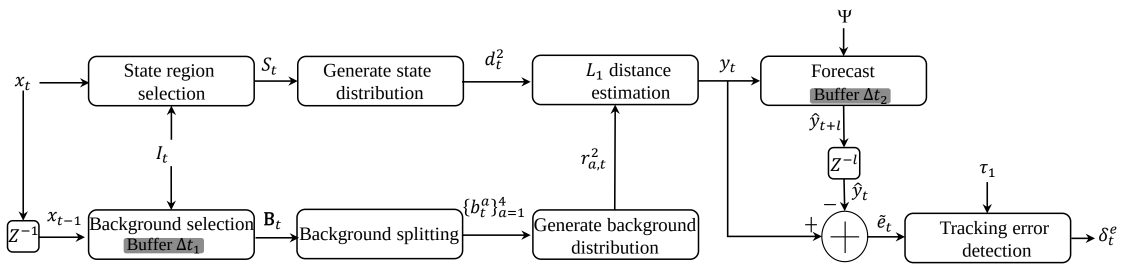


Figure 3.1: Block diagram of the proposed framework.

has a uniform range of values for any video data. We then detect significant changes (tracking failures) within the forecast error signal using an experimentally derived threshold. The method only uses tracking outputs x_t^k as input to evaluate performance and detect failures, thus making the approach generic to all trackers. We therefore remove the k superscript for the remainder of this chapter. Figure 3.1 shows the block diagram of the proposed approach.

The framework of Detecting Tracking Failure using Forecasting (DTFF) is presented in this chapter. The method for evaluating the performance of a tracker is presented in Section 3.2, while the forecasting and failure detection methods are covered in Section 3.3. Experimental setup, results and analysis for both failure detection and forecasting are covered in Section 3.4. Finally, Section 3.5 summarizes the chapter.

3.2 Tracking performance evaluation

Using the estimated target state x_t at each frame I_t, we evaluate tracking performance by performing background analysis. Background analysis is generally employed in *foreground detection* [21] and *tracking-by-detection* [6, 57, 58] approaches. *Foreground detection* approaches generate over time a background model to separate moving objects from the scene [21]. Background subtraction techniques have been employed as the simplest way to model the background, while more recent approaches employ statistical, cluster or estimation models [21]. *Tracking-by-detection* approaches track the target by employing local search regions around the estimated state from the current time instant [6, 45], typically using a sliding window approach [58]. This local region can either be circular with a certain radius [6, 58] or rectangular [45, 75, 99] with the size of rectangular region being larger than the target size. However, using a circular shape [6, 58] as the search region can limit the methods to sequences where object size is larger than the circle radius.

We employ a *tracking-by-detection* approach and extract a larger background region around the target state. Then, we generate distributions of both the state and background region using colour DF [146] and compare the two distributions to generate the tracking performance score. In the following subsections, we first explain the method for background region selection that is followed by an overview of colour DFs and their use in generating the distributions. The final subsection explains how the tracking performance score is computed.

3.2.1 Background selection

Using past motion and position information of the state over a sliding temporal window, we select in I_t the background region $\mathbf{B}_t$ around the state region S_t defined by x_t. $\mathbf{B}_t$ encloses both the state and its surrounding background.

To select $\mathbf{B}_t$, we first predict the target position in the next frame. We use past information over a short sliding temporal window, Δt_1, to determine the average displacement and the direction of movement of the target (Figure 3.2(a)). Let $\vec{\Omega}_{\Delta t_1}$ be the directional feature of a tracker [3], which represents both the average displacement and direction of the target over Δt_1 and is computed as:

$$\vec{\Omega}_{\Delta t_1} = \frac{1}{\Delta t_1} \sum_{t'=t-\Delta t_1}^{t-1} [u_{t'+1} - u_{t'}, v_{t'+1} - v_{t'}], \tag{3.1}$$

where the position of the target in I_t is predicted as $[\hat{u}_t, \hat{v}_t] = [u_{t-1}, v_{t-1}] + \vec{\Omega}_{\Delta t_1}$. The background region $\mathbf{B}_t$, centred at $[\hat{u}_t, \hat{v}_t]$ with width and height of $\hat{w}_t = w_{t-1} + \lceil (w_{t-1} + h_{t-1})/4 \rceil$ and $\hat{h}_t = w_{t-1} + \lceil (w_{t-1} + h_{t-1})/4 \rceil$ [45], respectively, where $\lceil . \rceil$ defines the ceil operation, is then selected from frame I_t.

3.2.2 Distribution Fields

DFs have been employed for background subtraction [40, 128] as well as for tracking [46, 100, 122, 146], and combine the power of histograms and intensity gradients to preserve both visual information and the spatial structure of the image region [146]. Composed of several layers (bins), a DF is a collection of probability distributions where each distribution defines the probability of a pixel taking the feature value (e.g. color intensity). Having dimensionality of 2+C, a DF is represented as a matrix where the first two dimensions represent the width and height of the image and while the feature space is indexed via the other C dimensions [122]. For example, using the gray-scale intensity as the feature space yields a 3-dimensional DF. We determine the distributions of the two regions using colour DF [146]. Using the RGB feature space results in a 4-dimensional DF, one for each channel of the image region.

Let $d_t(i,j,c,m)$ be the DF, where $i = 1,...,h_t$ and $j = 1,...,w_t$ define the pixel location in S_t, $c \in \{\text{R, G, B}\}$ is the feature channel and $m = 1,...,M$ is the index of the layer. A DF is generated in three steps. The first step explodes the image into multiple layers (Figure 3.2(c)) resulting in a Kronecker delta function at each pixel location:

$$d_t(i,j,c,m) = \begin{cases} 1 & \text{if} \quad \lceil \frac{S_t(i,j,c)}{\kappa} \rceil = M, \\ \\ 0 & \text{otherwise}, \end{cases} \tag{3.2}$$

where κ, the size of each layer, is the ratio between the maximum feature value (e.g. 255 for an RGB channel) and M. Although in a larger representation, $d_t(i,j,c,m)$ contains the same information as contained in the original image (S_t). The second step spatially spreads the information in $d_t(i,j,c,m)$ by convolving $d_t(i,j,c,m)$ with a 2-D Gaussian kernel h_{σ_1} over m as:

$$d_t^1(m) = d_t(m) * h_{\sigma_1}, \tag{3.3}$$

where σ_1 is the standard deviation of h_{σ_1} and $*$ is the convolution operation. Finally, to better handle small variations in brightness and subpixel motion [122] $d_t^1(m)$ is convolved with a 1-D Gaussian kernel h_{σ_2} (with standard deviation σ_2) over (i,j,c) (Figure 3.2(d)) as:

$$d_t^2(i,j,c) = d_t^1(i,j,c) * h_{\sigma_2}. \tag{3.4}$$

To compare the DF of $\mathbf{B}_t$ with $d_t^2(i,j,c,m)$, we divide $\mathbf{B}_t$ into four smaller and equally sized regions b_t^a, where $a = 1,...,4$, having width w_t and height h_t (Figure 3.2(b)). Then, using the same feature space, a distribution for each b_t^a ($r_{a,t}^2(i,j,c,m)$) is computed using Eqs. (3.2) - (3.4), where S_t, d_t, d_t^1 and d_t^2 are replaced by b_t^a, $r_{a,t}$, $r_{a,t}^1$ and $r_{a,t}^2$, respectively.

3.2.3 Tracking performance score

The two distributions can be compared using different distance measures such as the L_1, Bhattacharya or Mahalanobis distance. Since L_1 computes the difference at each pixel location (i,j), it can provide higher discrimination between the two distributions. We use the L_1 distance to

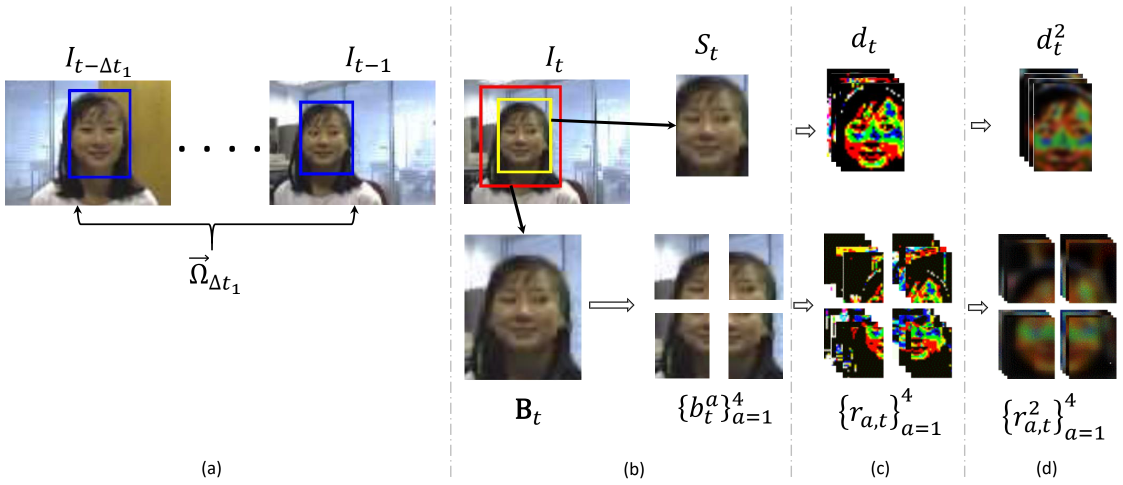


Figure 3.2: Background and state region selection. (a) $x_{t-\Delta t}$, ..., x_{t-1} (enclosed in the blue bounding boxes) and motion information $\vec{\Omega}_{\Delta t_1}$ over a sliding temporal window Δt_1; (b) background region $\mathbf{B}_t$ (enclosed in the red bounding box) and state region S_t (enclosed in the yellow bounding box) are selected from frame I_t and $\mathbf{B}_t$ is further split into smaller regions $\{b_t^a\}_{a=1}^4$; (c)-(d) distributions of $\mathbf{B}_t$ and S_t represented with colour DF [146].

compare $d_t^2(i,j,c,m)$ and $r_{a,t}^2(i,j,c,m)$ on each channel as:

$$y_t^a = \frac{1}{w_t\, h_t\, M} \sum_{c\in\{\mathrm{R,G,B}\}} \left(\alpha_c \sum_{i=1}^{h_t} \sum_{j=1}^{w_t} \sum_{m=1}^{M} \left| d_t^2(i,j,m) - r_{a,t}^2(i,j,m) \right| \right), \tag{3.5}$$

where $|.|$ denotes the absolute value. We normalise the distance by the height and width of the state and the number of layers. The weight α_c assigned to each channel is computed as:

$$\alpha_c = \frac{\left| \mu_{S_t}^c - \mu_{b_t^a}^c \right|}{\sum\limits_{c\in\{\mathrm{R,G,B}\}} \left| \mu_{S_t}^c - \mu_{b_t^a}^c \right|}, \tag{3.6}$$

where $\mu_{S_t}^c$ and $\mu_{b_t^a}^c$ are the mean R, G, B values for S_t and b_t^a, respectively. Weighting the colour channels allows us to exploit the most discriminative one(s) when comparing the two distributions.

The overall tracking performance score y_t is determined by quantifying the similarity between $\mathbf{B}_t$ and S_t as:

$$y_t = \frac{1}{4} \sum_{a=1}^{4} y_t^a, \tag{3.7}$$

where low (high) values of y_t indicate similarity (dissimilarity) between $\mathbf{B}_t$ and S_t.

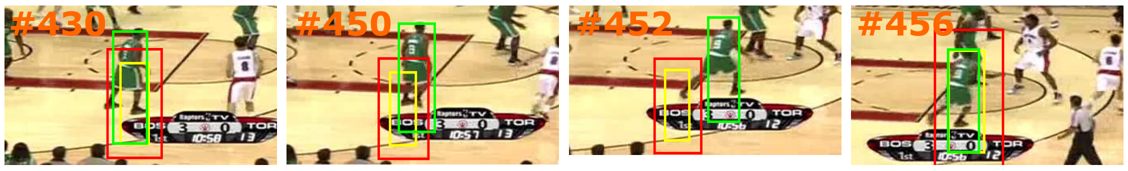
3.3 Detecting tracking failure

We detect tracking failure by employing time series analysis to model $\mathbf{Y} = \{y_t\}_{t=1}^{T}$ as a univariate discrete time series for forecasting. y_t can only estimate the tracking performance at each t, where tracking failures and noise generate significant and small changes, respectively in $\mathbf{Y}$. The noise can be generated by noisy video data and therefore the range of $\mathbf{Y}$ can vary for different video data. We model $\mathbf{Y}$ using a forecasting model to minimize noise and to generate a signal that has a uniform range of values for all video data. In the following section we discuss the forecasting and the failure detection methods.

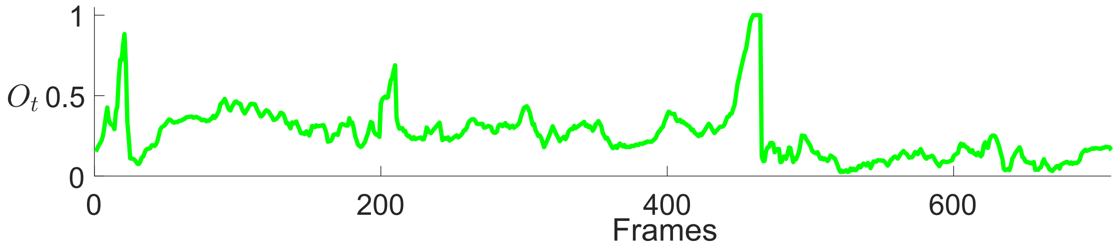
3.3.1 Forecasting

Forecasting has been employed for multiple applications within the domain of economics [136], environment [70], medicine, politics and other fields [131, 147]. It deals with the prediction of future events either in the short-term or the long-term period. A forecasting model makes use of the past information and a forecasting model, where the model is used to determine the patterns within the time series and the relationships between the past and current values. Then, using the estimated parameters and the forecasting model, future values of the time series are predicted over the *forecast lead time*. Finally, a difference between the original time series and its forecast generates the *forecast error* signal. Forecasting methods such as moving average models [63] have flat forecast functions and generally do not take past information into account. ARMA models [22] are built using past data and forecast using both past and present data. State-space models, such as the KF, require the model of the time series to be known beforehand for forecasting [50]. SVM [119] and neural network models [50] are more complex than ARMA for forecasting. A comprehensive survey of time series forecasting is presented in [50]. We employ ARMA to model $\mathbf{Y}$, where the difference between the forecast and the original returns a re-scaled signal, highlighting only the significant changes (tracking failure).

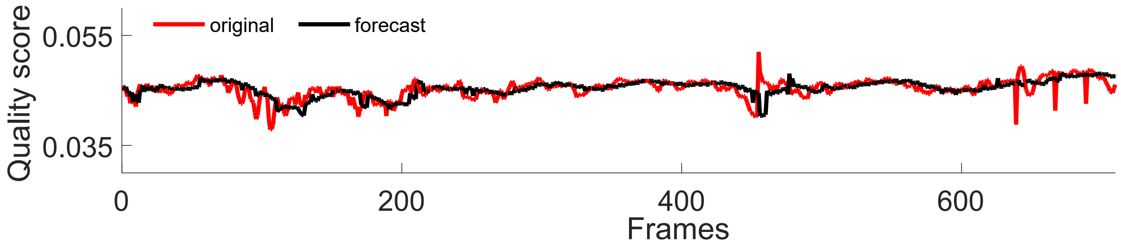
We next describe the general ARMA model and the forecasting approach which is then used for detecting tracking failure. ARMA(p,q) models are defined by their auto regressive (AR) and moving average (MA) orders $p = 0,...,P$ and $q = 0,...,Q$, respectively. Determining the right model requires identification of P and Q that can be achieved by simpler methods such as by visually inspecting the behavior of the Auto Correlation Function (ACF) and Partial Auto Correlation Function (PACF) plots of $\mathbf{Y}$ or by employing statistical tests such as the Akaike and



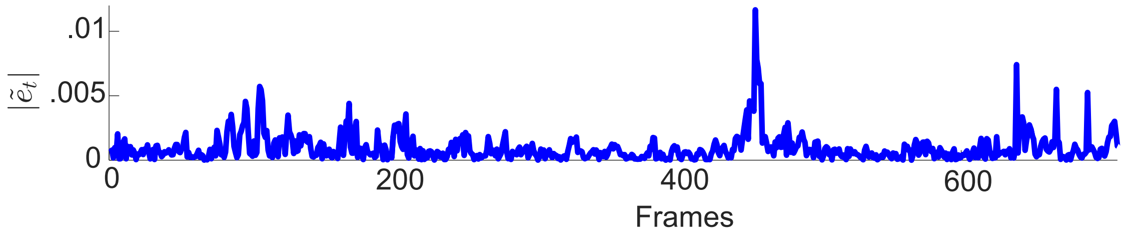
(a)



(b)



(c)



(d)

Figure 3.3: Example for track performance evaluation with forecasting using the results produced by tracker DSST [34, 36] on the VOT2014 sequence *basketball*. The tracker is re-initialised after a failure at frame 452. (a): Tracking results. Red, yellow and green represent $\mathbf{B}_t$, S_t and the ground-truth state, respectively. (b) Tracker performance measured as O_t (Section 3.4.2); (c) original y_t and its forecast $\hat{y}_{t+l}$; (d) forecast error signal $|\tilde{e}_t|$. The tracking failure at frame 452 is reflected as a significant change within $|\tilde{e}_t|$.

Bayesian Information Criteria [50]. The behavior of the ACF and PACF plots of a time series can help determine the values of P and Q of the ARMA model. The ACF of an AR(p) process goes to zero at an exponential rate (tails-off), while the PACF cuts off (becomes zero) after order P. Conversely, the ACF of an MA(q) process cuts off after order Q, while the PACF tails-off [22].

The general form of an ARMA(p,q) model is given as:

$$y_t = \sum_{p=0}^{P} \phi(p)\, y_{t-p} + \sum_{q=0}^{Q} \vartheta(q)\, \varepsilon_{t-q} + \beta, \tag{3.8}$$

where, β is a constant; $\phi(p)$ and $\vartheta(q)$ are the AR and MA polynomials, with orders p and q respectively, and $\varepsilon_t = \hat{y}_t - y_t$ is termed as the residual (estimation error) [22].

Using P, Q and the past values of y_t within a sliding temporal window Δt_2, $\hat{y}_t$: the estimated values (obtained by fitting the forecast model to y_t) are determined using Eq. (3.8) by replacing y_t, $\phi(p)$, $\vartheta(q)$ and β, with the estimated parameters $\hat{y}_t$, $\hat{\phi}(p)$, $\hat{\vartheta}(q)$ and $\hat{\beta}$, respectively. The parameters ($\hat{\phi}(p)$, $\hat{\vartheta}(q)$ and $\hat{\beta}$) can be recursively estimated using techniques such as the conditional least squares or the maximum likelihood methods [22]. Using the parameters $\Psi = \left\{P, Q, \hat{\phi}(p), \hat{\vartheta}(q), \hat{\beta}\right\}$ forecasts are recursively computed over the forecast lead time $l \geq 1$ at time t as:

$$\hat{y}_{t+l} = \begin{cases} \sum_{p=0}^{P} \hat{\phi}(p)\, \hat{y}_{t+l-p} + \sum_{q=0}^{Q} \hat{\vartheta}(q)\, \varepsilon_{t+l-q} + \hat{\beta} & \text{for} \quad l < Q, \\ \\ \sum_{p=0}^{P} \hat{\phi}(p)\, \hat{y}_{t+l-p} + \hat{\beta} & \text{otherwise,} \end{cases} \tag{3.9}$$

where ε_t is replaced by zeros for $l > Q$, because it has not occurred yet [22]. The forecasting error $\tilde{e}_{t+l} = y_{t+l} - \hat{y}_{t+l}$ determines the accuracy of a forecasting approach [50]: low (high) values indicate good (bad) forecasts. Note that the estimation error (ε_t): difference between the original (y_t) and the estimated values ($\hat{y}_t$) is different from the forecasting error ($\tilde{e}_{t+l}$): difference between the original (y_t) and forecasts ($\hat{y}_{t+l}$) [95].

3.3.2 Tracking failure detection

We then determine time instants of tracking failure by employing the absolute values of forecast error $|\tilde{e}_{t+l}|$. Since the values of $\hat{y}_{t+l}$ are dependent on past values of y_t between $t - \Delta t_2$ and t, $|\tilde{e}_{t+l}|$ temporally smooths y_t. A tracking failure is represented by a significant change in the value of y_t that is then detected by e_{t+l} as:

$$e_{t+l} = \begin{cases} 1 & \text{if} \, |\tilde{e}_{t+l}| \geq \tau_1, \\ \\ 0 & \text{otherwise,} \end{cases} \tag{3.10}$$

where the threshold τ_1 is determined experimentally. Finally, if $\delta_t^e = e_{t+l} - e_{t+l-1}$, then $\delta_t^e = 1$ indicates when a tracking failure first occurs.

3.4 Experimental results and analysis

This section presents an experimental analysis of the DTFF framework. We employ a heterogeneous dataset for testing purposes to evaluate the efficiency of DTFF in detecting time instants when the tracker fails. Using the ST tracker (Section 2.2.5), we test the method over the D1 dataset and selected sequences from the Object Tracking Benchmark (OTB)[1] dataset (Appendix A). Finally, to demonstrate the flexibility of the DTFF framework in detecting tracking failures, we test the approach by using tracking results from four different trackers: DSST, SAMF, KCF, PLT_14 over the Visual Object Tracking (VOT) dataset from the VOT2014 challenge[2]. Further details and target initializations of the datasets are presented in Appendix A.

First, we present the details of the experimental parameters, the evaluation measures and the SOA methods used within the framework in Section 3.4.1, Section 3.4.2 and Section 3.4.3, respectively. Experimental results and analysis for tracking failure detection over the OTB and VOT datasets are presented in Section 3.4.4 and 3.4.5, respectively. Finally, we present results of forecasting accuracy in Section 3.4.6.

3.4.1 Experimental parameters

Model building and parameter estimation of the framework is achieved using the MATLAB built-in *arima* and *estimate* functions, respectively. As finding the best fit for $\mathbf{Y}$ is out of the scope of this work, we choose the values of $P = Q = 1$ by visually inspecting the ACF and PACF plots of $\mathbf{Y}$ based on their properties as explained in Section 3.3.1. The temporal window $\Delta t_1 = 10$ provides an optimal value to encode the average target displacement and direction. The standard deviation of the 2-D Gaussian kernel is set to $\sigma_1 = 1$ and 2 for the u and v directions, respectively, and $\sigma_2 = 0.625$ for the 1-D Gaussian kernel as in [122], while the number of layers, $M = 32$ provides a better discrimination between background and target distributions. Using the dataset D1 we perform an empirical analysis to determine the optimal values of Δt_2, l and τ_1. For determining the optimal amount of past data required to build the forecast model we change Δt_2 values as 10 and 20. The forecast lead time is varied as l = 5, 10, 25, 50 to determine the performance for both short-term and long-term forecasts. Finally, the threshold τ_1 was varied between 0.003 and 0.009 with a step size 0.001. Based on the results obtained on dataset D1, we find that the best results were obtained by using $\Delta t_2 = 20$, $l = 5$ and $\tau_1 = 0.004$. Using these values, we further test

[1]http://cvlab.hanyang.ac.kr/tracker_benchmark/datasets.html

[2]http://www.votchallenge.net/vot2014/index.html

the approach over both OTB and VOT datasets (Appendix A).

3.4.2 Evaluation measures

In order to measure tracking performance, we measure the deviation from the GT as the overlap score O_t between the tracker output x_t and GT annotation as [114]:

$$O_t = 1 - \frac{2|A_t^x \bigcap A_t^{\mathrm{GT}}|}{|A_t^x| + |A_t^{\mathrm{GT}}|}, \tag{3.11}$$

where A_t^x and A_t^{GT} represent the area in pixels of the estimated, x_t, and GT target locations, respectively; $|A_t^x \bigcap A_t^{GT}|$ is their spatial overlap in pixels. $O_t \in [0,1]$ and values close to 0 (1) indicate high (low) tracking performance.

For measuring the performance of DTFF in detecting tracking failures, time instants when O_t changes from success ($O_t < 1$) to failure ($O_t = 1$) are determined by the GT transitions, ($\delta_t^O = O'_t - O'_{t-1}$, if $O'_{t-1} = 0$ and $O'_t = 1$, otherwise $\delta_t^O = 0$), where O'_t is determined as [114]:

$$O'_t = \begin{cases} 1 & \text{if} \quad O_t = 1, \\ \\ 0 & \text{if} \quad O_t < 1. \end{cases} \tag{3.12}$$

We use the number of true positives (n_{TP}), false positives (n_{FP}), false negatives (n_{FN}) and true negatives (n_{TN}) to measure the performance of DTFF in detecting tracking failures. For DTFF n_{TP} (n_{FN}) indicate whether the decisions, δ_t^e, of the proposed method correspond correctly (incorrectly) to the tracking failure decisions generated by δ_t^O. Similarly, a correct (incorrect) match of the tracking success decisions between δ_t^e and δ_t^O is determined by n_{TN} (n_{FP}). A tolerance window of ± 5 frames is used to match δ_t^e with each δ_t^O. Using n_{TP}, n_{FP}, n_{TN} and n_{FN}, we then compute the precision, $P = \frac{n_{TP}}{(n_{TP}+n_{FP})}$, the recall, $R = \frac{n_{TP}}{(n_{TP}+n_{FN})}$, the F-score $= 2.\frac{P.R}{P+R}$, and the false positive rate $FPR = \frac{n_{FP}}{n_{FP}+n_{TN}}$ [125]. Values for the F-score close to 1 (0) indicate high (low) accuracy.

For evaluating the accuracy of the forecasting model employed within DTFF, the forecast error $\tilde{e}_t$ (Section 3.3) and its root mean squared error [65] are the most common measures. However due to their scale dependence, we employ a scale independent measure known as the mean absolute scaled error MASE [65]:

$$MASE = \frac{1}{T}\sum_{t=1}^{T}\left(\left|\frac{\tilde{e}_t}{\frac{1}{T-1}\sum_{t=2}^{T}|y_t - y_{t-1}|}\right|\right). \tag{3.13}$$

We use the mean absolute percentage error MAPE = $\frac{1}{T}\sum_{t=1}^{T}(|\tilde{e}_t/y_t| \cdot 100)$ to compute the percentage error. High (low) values of both MAPE and MASE indicate high (low) forecasting accuracy.

DTFF employs a forecasting model to smooth out values of the tracking performance score y_t, giving DTFF the advantage to be flexible over different datasets and trackers. Let a generic z_t represent $|\tilde{e}_t|$, y_t or the tracking performance scores generated by the SOA methods, and $\mathbf{Z} = \{z_t\}_{t=1}^{T}$ be the corresponding time-series. We normalise each z_t as:

$$z'_t = \frac{z_t - \min(\mathbf{Z})}{\max(\mathbf{Z}) - \min(\mathbf{Z})}, \tag{3.14}$$

to enable the comparison of variations of the corresponding values over the whole dataset.

3.4.3 Selected benchmark methods

We compare DTFF for detecting tracking failure with two variations of the proposed approach: Raw and Naive; one SOA method for tracker failure detection: Covariance Features (CovF) [113] and two SOA feature descriptors employed for video tracking: RGB Histograms (RgbHist) and RGB+LBP Histograms (RLHist) [101].

For Naive, we change the forecasting model to Naive [50] that detects tracking failures by forecasting values of y_t equal to the last observed value ($\hat{y}_{t+l} = y_t$). The threshold value for the method is set by varying the threshold values between 0.002 and 0.007 with a step size of 0.001 over D1 where the threshold value = 0.004 returns the best results and is selected for testing. For Raw, we remove the ARMA forecasting model and use raw y_t values. Value of the threshold = 0.039 for failure detection is determined varying the values of the threshold between 0.025 and 0.045 with a step size of 0.001 over dataset D1.

We replace our DF based feature with other SOA features i.e. CovF [113], RgbHist and RLHist [101] for failure detection within our state-background discrimination framework. CovF [113] employs a 5-dimensional target descriptor based on the colour and position values to determine tracking performance. For failure detection, the threshold for CovF is set to 2.3 as used in [113]. RgbHist generates histograms based on the colour feature (RGB), while RLHist is based on his-

Table 3.1: Comparison of tracking failure detection performance in terms of precision (P), recall (R), F-score (F) and false positive rate (FPR). The results are presented as total values over the OTB dataset. The best results are indicated by bold font. The last row shows the mean±standard deviation of z'_t. Key — DTFF: Detecting Tracking Failure using Forecasting; NAIVE: failure detection by forecasting y_t via the Naive forecasting model [50]; RAW: failure detection using raw y_t values; CovF: Covariance Features [113]; RgbHist: RGB Histogram [101]; RLHist: RGB+LBP Histogram [101].

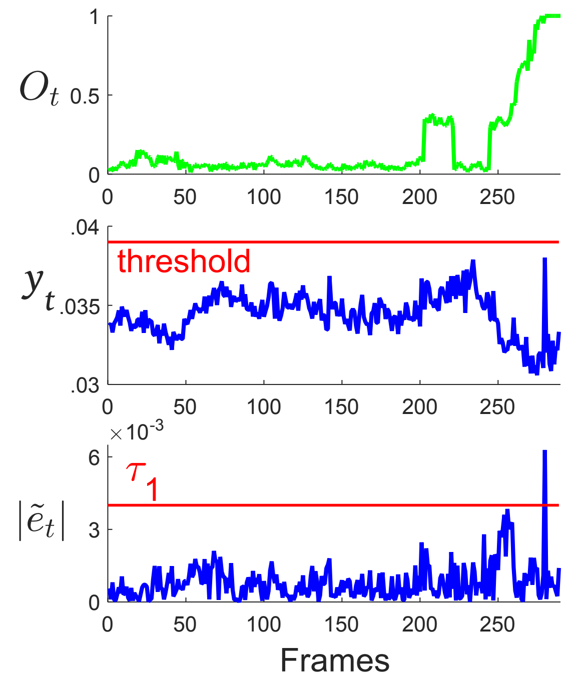
	DTFF	NAIVE	RAW	CovF	RgbHist	RLHist
P	.110	.111	**.122**	.087	.083	.078
R	**.714**	.667	.405	**.714**	.595	.667
F	**.191**	.190	.188	.155	.146	.140
FPR	.037	.035	**.019**	.048	.042	.051
$\mu \pm \sigma$	.17±.18	.15±.17	.60±.23	.30±.17	.67±.16	.61±.16

tograms generated by combining both the RGB and local binary patterns (LBP) features. RgbHist and RLHist are employed to generate a tracking performance score and then the threshold = 0.88 value for both methods is selected by varying the threshold values between 0.80 and 1.00 with a step size of 0.01.

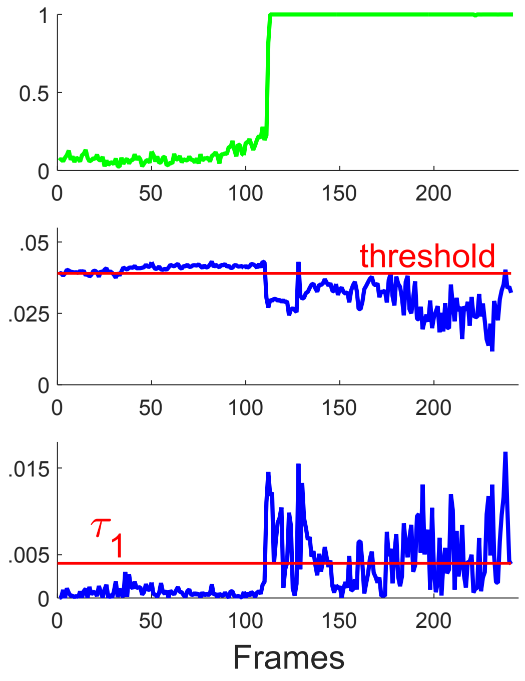
For DTFF, NAIVE and CovF the tracking failure is detected for values above their respective thresholds, while for RAW, RgbHist and RLHist for values below their respective thresholds.

3.4.4 Tracking failure detection

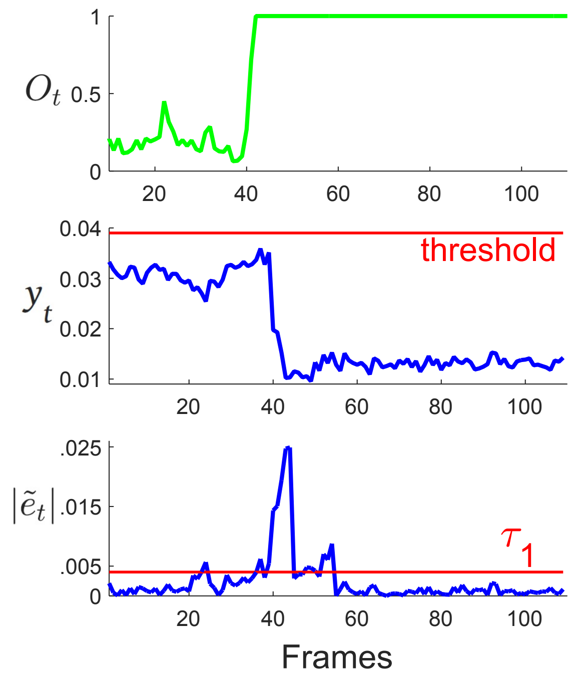
We first compare DTFF with NAIVE and RAW, and then with CovF, RgbHist and RLHist on the OTB dataset. While the values of y_t vary across sequences (see Figure 3.4(a)-(d)), forecasting enables us to generate a signal with the same range of values for the whole dataset. In *CarDark* an illumination variation and background clutter cause a tracking failure between frames 270 and 280, while in *Crossing* a tracking failure occurs due to scale variations. RAW achieves lower P than DTFF and NAIVE (Table 3.1), which detect these failures with their respective forecasting approaches. RAW achieves a lower FPR and hence a better P than DTFF and NAIVE, because for sequences where the tracker fails to re-acquire the target after a failure, the values of y_t fall below the threshold. However, since the tracker is not stopped (or re-initialised), y_t generates false significant changes, which are recorded as tracking failures by DTFF and NAIVE (see Figure 3.4(b) and (d)). Since NAIVE forecasts values at time t for the complete forecast lead time (l), it may suppress some of the false significant changes (false positives) of y_t resulting in a better P than DTFF. However, this behaviour also results in a lower R for NAIVE. The R of DTFF outperforms that of RAW and NAIVE by 76% and 7%, respectively, hence DTFF achieves a better F-score.



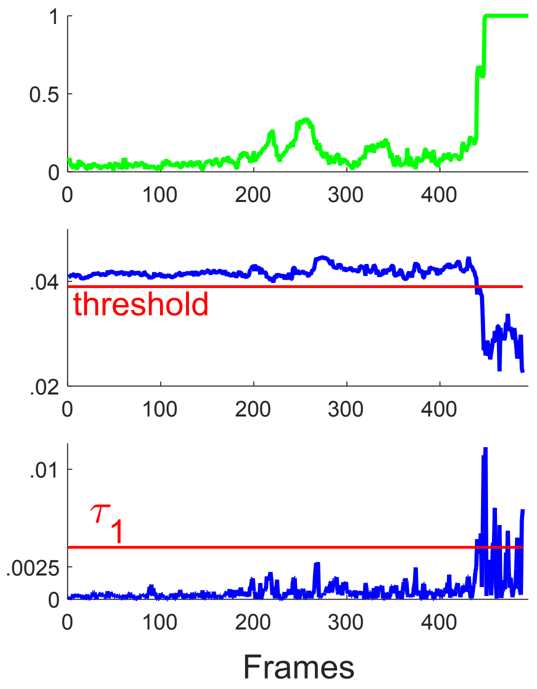
(a)



(b)



(c)



(d)

Figure 3.4: Example of variation of the tracking performance score, y_t, for (a) *CarDark*, (b) *CarScale*, (c) *Crossing*, (d) *Doll*. Top row: tracker performance measured as O_t. Middle row: tracking performance score y_t (blue line). Bottom row: forecast error $|\tilde{e}_t|$ (blue line). τ_1 and 'threshold' are used to detect time instants when the tracker fails for DTFF and `RAW`, respectively.

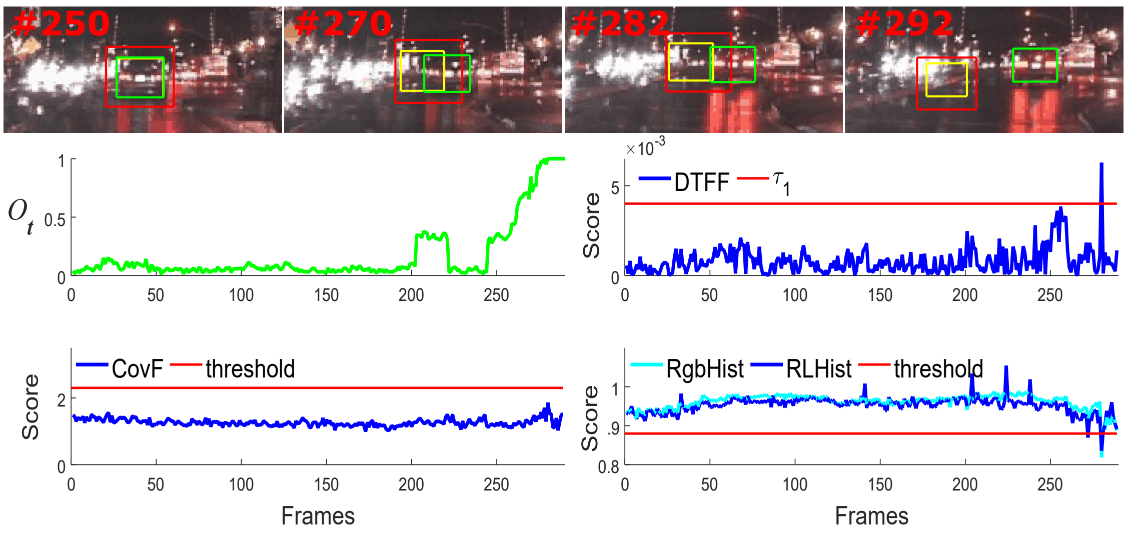


Figure 3.5: Example of tracking performance scores for *CarDark*. First row: Sample tracking results: the red, yellow and green bounding boxes represent $\mathbf{B}_t$, S_t and the ground-truth state, respectively. Second row: tracker performance measured as O_t (left) and $|\tilde{e}_{t+l}|$ score measured with DTFF (right). Third row: tracking performance score measured with CovF (left); and tracking performance score measured with RgbHist and RLHist (right).

DTFF outperforms CovF, RgbHist and RLHist in terms of F-score, giving an overall improvement of 23%, 31% and 36% compared to CovF, RgbHist and RLHist, respectively. CovF, RgbHist and RLHist have a variable range of values over the dataset that leads to false positive and false negative tracking failure decisions. Although CovF detects the same number of tracking failures as DTFF, a lower P results in lower F-score. RgbHist and RLHist are affected by these variations in terms of R. Furthermore, CovF, RgbHist and RLHist have similar failure modes to DTFF since their descriptors use colour as their primary feature. CovF achieves better results than RgbHist and RLHist, possibly because it uses position information as well, while RLHist improves over RgbHist, possibly due to the additional LBP features. Sample results for sequence *CarDark* and *Doll* are shown in Figure 3.5 and Figure 3.6, respectively.

DTFF generates false positive tracking failure decisions when RAW generates significant changes after a tracker has failed. Furthermore, DTFF fails to detect tracking failures when RAW does not generate a change in y_t due to background clutter (in *Liquor* and *MountianBike*), occlusions (in *Jogging*), sudden background lighting changes (in *Singer1*) and fast target rotation and background clutter (in *MotorRolling*).

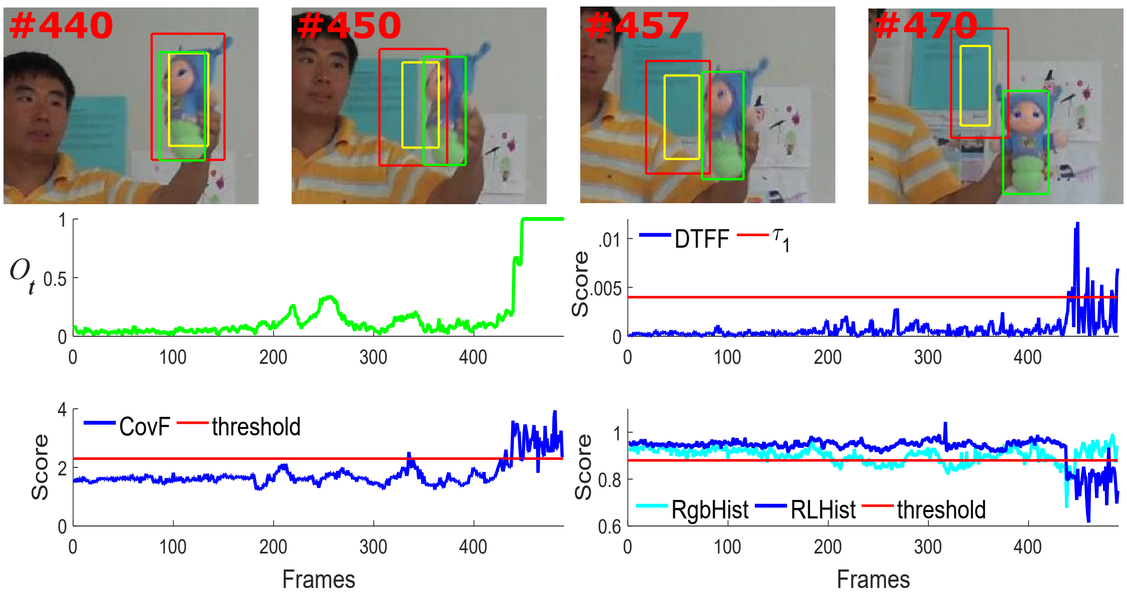


Figure 3.6: Example of tracking performance scores for *Doll*. First row: Sample tracking results: the red, yellow and green bounding boxes represent $\mathbf{B}_t$, S_t and the ground-truth state, respectively. Second row: tracker performance measured as O_t (left) and $|\tilde{e}_{t+l}|$ score measured with DTFF (right). Third row: tracking performance score measured with CovF (left); and tracking performance score measured with RgbHist and RLHist (right).

3.4.5 Results on the Visual Object Tracking dataset

Finally, we analyse the flexibility of DTFF via an experimental comparison with other methods using results from four trackers (DSST [34, 36], SAMF [81], KCF [62], PLT_14 [61]) and sequences from the VOT2014 challenge [73] (see Table 3.2). Note that VOT re-initialises trackers after failure (O_t=1): the tracker is stopped for the subsequent five frames and then is re-initialised with GT. In order to compensate for the missing tracking results, we keep for these five frames the same tracking result obtained when the tracker fails. The re-initialisation of the trackers allows DTFF to reduce its FPR and to achive a better F-score than RAW. Overall, DTFF improves by 51% and 94% in terms of F-score over both RAW and NAIVE, respectively. Using the forecast error signal allows DTFF to detect tracking failures that are not detected by RAW. DTFF detects more tracking failures than CovF, RgbHist and RLHist, and achieve the best R values for all the four trackers and a better F-score for trackers DSST and PLT_14, indicating that the threshold, τ_1, is applicabile to various datasets. However, RgbHist and RLHist achieve a better F-score for KCF and SAMF, respectively, due to a smaller FPR. All the approaches achieve their best results for DSST followed by KCF, SAMF and PLT_14.

Furthermore, we display results with three decimal digits to better compare the performance of the methods. Rounding to two decimal digits converts most FPR results equal. For e.g. in

Table 3.2: Comparison of tracking failure detection performance in terms of precision (P), recall (R), F-score (F) and false positive rate (FPR). The results are presented as total values over the whole VOT2014 dataset. The best results are indicated by bold font. The last row for each tracker shows the mean$\pm$standard deviation of z'_t. Key — DSST: Discriminative Scale Space Tracker [34, 36]; KCF: Kernelized Correlation Filter [62]; SAMF: Scale Adaptive KCF tracker [81]; PLT_14: Pixel based LUT Tracker [61]; DTFF: Detecting Tracking Failure using Forecasting; NAIVE: failure detection by forecasting y_t via the Naive forecasting model [50]; RAW: failure detection using raw y_t values; CovF: Covariance Features [113]; RgbHist: RGB Histogram [101]; RLHist: RGB+LBP Histogram [101].

	DSST						SAMF					
	DTFF	NAIVE	RAW	CovF	RgbHist	RLHist	DTFF	NAIVE	RAW	CovF	RgbHist	RLHist
P	**.154**	.124	.080	.077	.136	.135	.118	.078	.053	.059	.106	**.126**
R	**.700**	.326	.395	.465	.488	.651	**.488**	.220	.244	.342	.415	**.488**
F	**.252**	.180	.135	.132	.213	.224	.190	.115	.087	.100	.168	**.200**
FPR	.016	**.010**	.020	.024	.013	.018	.015	**.011**	.018	.022	.014	.014
$\mu\pm\sigma$	.18±.17	.16±.16	.54±.19	.35±.17	.66±.16	.62±.14	.18±.17	.16±.16	.54±.19	.34±.17	.64±.16	.63±.13
	KCF						PLT_14					
	DTFF	NAIVE	RAW	CovF	RgbHist	RLHist	DTFF	NAIVE	RAW	CovF	RgbHist	RLHist
P	.158	.095	.070	.093	**.183**	.145	**.065**	.057	.041	.029	.016	.045
R	**.535**	.256	.302	.419	.488	.395	.435	.304	**.478**	.217	.130	.261
F	.243	.138	.114	.152	**.266**	.213	**.113**	.096	.076	.051	.028	.076
FPR	.012	.010	.017	.017	**.009**	.010	.014	**.012**	.025	.017	.019	.015
$\mu\pm\sigma$	.16±.16	.16±.16	.55±.18	.36±.17	.68±.15	.64±.13	.19±.17	.17±.16	.57±.19	.41±.17	.61±.17	.61±.15

Table 3.2 under the DSST tracker, the results for DTFF, RAW, CovF and RLHist round to 0.02 while for NAIVE and RgbHist to 0.01, making it difficult to compare performance. Using two decimal digits for P, R and F-score does not have the same effect, however, we use three decimal digits to keep a uniformity of results. On the contrary using four instead of three does not cause any significant difference between the results.

3.4.6 Forecast accuracy

Results over the OTB dataset for forecasting accuracy are presented in Fig. 3.7. The results indicate that with increasing forecast lead time (l), forecasting accuracy decreases. Furthermore, the results also indicate that NAIVE is able to achieve overall best results for both Δt_2=10 and 20. The scaling factor in MASE Eq. 3.13 (based on naive forecast model), provides Naive an upper edge on other forecasting models. Furthermore, Naive uses the original values (y_t) for the entire l allowing it achieve better forecast accuracy. However, Naive is generally employed as a benchmark against other forecasting models and not considered for forecasting since it may not efficiently utilize past information [1]. Comparing DTFF (ARMA) with Drift and EWMA for longer l, EWMA overall achieves better results. However, since the proposed approach performs better with shorter l, we see that DTFF achieves better results for both MAPE and MASE for

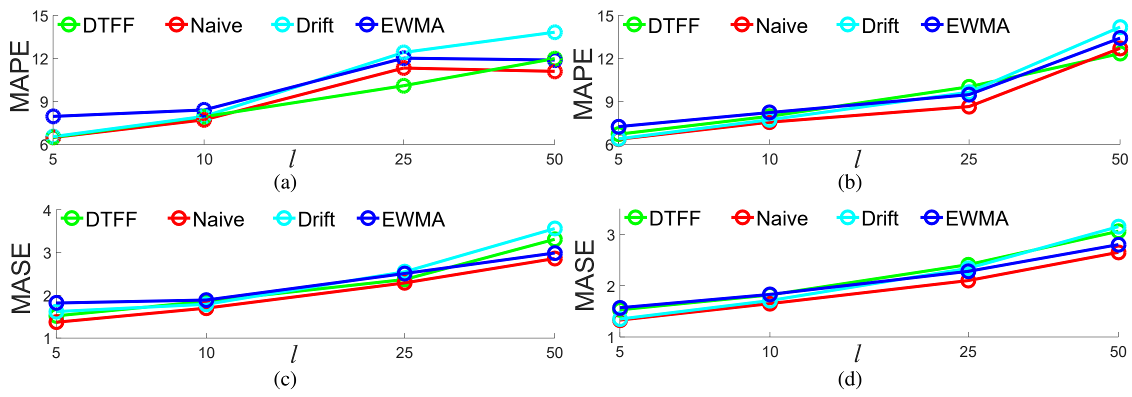


Figure 3.7: Comparison of forecast accuracy for selected forecasting approaches measured as mean absolute percentage error (MAPE) and mean absolute scaled error (MASE) for varying forecast lead times l. MAPE for (a) $\Delta t_2 = 10$ (b) $\Delta t_2 = 20$; MASE for (c) $\Delta t_2 = 10$ (d) $\Delta t_2 = 20$.

shorter l. In summary, simple forecasting models that do not extrapolate minor or major changes, may in some cases outperform better models that identify and extrapolate these changes in the time series data.

3.5 Summary

This chapter presented a formulation for detecting tracking failures using a forecasting model. First, we evaluate the performance of the tracker by employing a state-background discrimination approach. This is achieved by determining distributions of both the state region and a larger background region around the target state. A comparison of the two distributions returns the tracking performance score. The score reflects the tracking failures as significant changes, while small changes reflect noise generated due to the noisy video data. To suppress noise and detect only the significant changes, we model the tracking performance score using a forecasting approach. We employ the ARMA forecasting model to generate short-term forecasts (future values) of the tracking performance score. Then, a difference between the original and forecast values generates the forecast error signal, that is used to detect tracking failures by employing an experimentally determined threshold. Values of the forecast error signal above the threshold reflect a tracking failure. We demonstrate the effectiveness of the method by testing it over two different datasets and five different trackers. We also perform experimental comparison with two variations of the proposed method and three SOA methods.

Furthermore, to weight each tracker during the fusion process (Section 4.5), we use the tracking performance score based on the state-background discrimination approach. Assigning performance based weights to tracker during the fusion process may help the fusion process in improving the overall tracking accuracy in comparison to the equal weighted fusion approach.

Chapter 4

Clustering based approach to tracker-level fusion

4.1 Introduction

Fusion for visual tracking can be performed at feature or tracker-level [151]. Feature-based approaches fuse multiple features in a single tracking framework to adapt to appearance changes [7, 43, 62, 127, 151]. Tracker-level fusion combines the output of multiple trackers either in a sequential (cascade) manner [96, 118, 124], using outputs from specific trackers [89, 168], employing likelihood-based fusion [74], in parallel [49] or with a correction framework [16]. Online performance evaluation of the trackers or features prior to fusion can help to increase tracking accuracy, where this performance evaluation identifies the best performing trackers or features at run time [116]. Performance can be estimated using features such as the filter uncertainty [16, 91] and likelihood [74] or by using properties of the trajectories such as target velocity [123] and correlation between tracking outputs [49].

In this chapter we present a tracker-level fusion framework that combines the outputs of selected trackers in parallel. A block diagram of the framework is presented in Figure 4.1. Using the outputs (x_t^k) of each tracker (F^k), we group the trackers hierarchically based on their agreement in estimating the target state in terms of spatial location and direction of movement over a short temporal window Δt_3 (*Tracklet correlation*). This agreement is quantified by the correlation scores ($R_{\Delta t_3}^{i,j}$) between pairs of trackers F^i and F^j. Using these spatio-temporal scores, we generate the partition hypothesis $\{\mathbb{P}_{p,t}\}_{p=1}^{K}$ to determine the groups (clusters) of trackers that are within the same region (*Partition generation*). After determining the best partition $\mathbb{P}_t^*$ (*Partition*

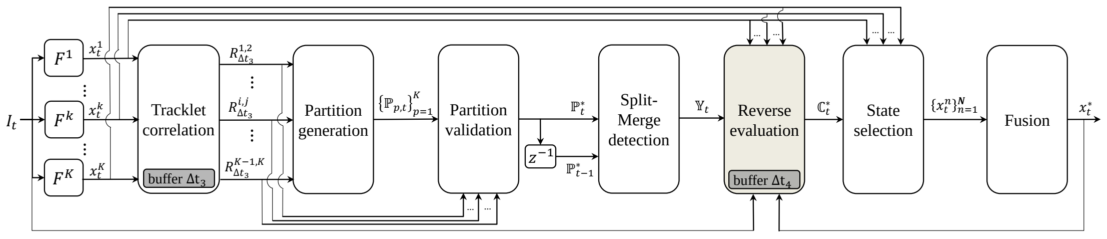


Figure 4.1: Block diagram of the proposed approach to fuse the output of K trackers.

validation), we identify the *on-target* (successfully following the target) cluster $\mathbb{C}_t^*$. This identification is achieved using an online performance evaluator (*Reverse evaluation*) that uses reverse-tracking [149] over a sliding temporal window Δt_4. Such evaluation requires standard tracking outputs (e.g. bounding boxes), thus providing a generic evaluator across all trackers. Finally, the outputs from the *on-target* cluster $\mathbb{C}_t^*$ are fused to estimate the final target state (x_t^*) (*Fusion*). $\mathbb{C}_t^*$ is propagated over time until a split or merge is detected (*Split-Merge detection*), which happens when trackers leave or join the cluster $\mathbb{C}_t^*$, respectively. The framework employs two temporal windows Δt_3 and Δt_4 (Figure 4.2), during which data is buffered from future and past time instants, respectively. The temporal window used for determining the spatio-temporal agreements makes the approach suitable for applications that can tolerate a short latency of Δt_3.

In this chapter, we first discuss tracker clustering method in Section 4.2, while the adaptive time-reversed evaluation method is presented in Section 4.3. The fusion approach is discussed in Section 4.4, which is followed by the experimental setup in Section 4.5. We then present the experimental analysis of the three sections: tracker clustering, adaptive time-reversed evaluation and fusion, in Sections 4.6, 4.7 and 4.8, respectively. Results and discussion on the computational time of the framework are covered in Section 4.9. Finally, Section 4.10 summarizes the chapter.

4.2 Tracker clustering

Assuming that all the trackers are initialized with ground truth, the proposed framework starts with a single cluster. However, over time trackers may fail due to the challenges involved in the visual data and split into different clusters. To determine the changing configuration and number of clusters, we measure the spatio-temporal agreement between pairs of trackers at each frame I_t. These agreements are used to determine the partition hypotheses dividing the K trackers into clusters and then to find the partition that best represents the spatio-temporal relationships among the trackers.

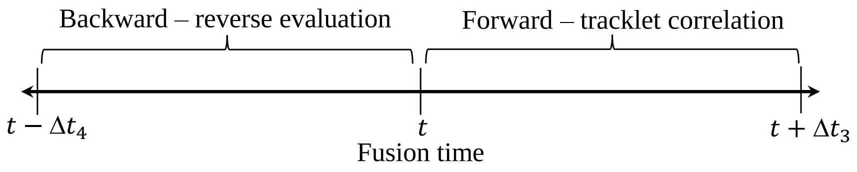


Figure 4.2: Temporal windows Δt_3 and Δt_4 employed by the proposed approach to account for forward and backward data, respectively. Forward data are used to determine the relationships among trackers via their trajectories (Section 4.2). Backward data are used to check tracker performance via a time-reversed evaluator (Section 4.3).

4.2.1 Tracklet correlation

We combine spatial and temporal features of the short-term trackers' trajectories (tracklets) to obtain a set of pair-wise correlation scores $R^{i,j}_{\Delta t_3}$, for $1 \leq i, j \leq K$ with $i \neq j$; for pairs of trackers F^i and F^j over a temporal window Δt_3. These scores quantify the agreement between F^i and F^j by determining the amount of spatial overlap and the temporal direction of movement using their tracking outputs x^i_t and x^j_t, respectively. The spatial agreement for F^i and F^j at frame I_t is determined as:

$$O^{i,j}_t = \frac{2|A^i_t \bigcap A^j_t|}{|A^i_t| + |A^j_t|}, \tag{4.1}$$

where A^i_t and A^j_t are the sets containing the pixels of the bounding boxes generated by tracker F^i and F^j, respectively, and $|.|$ is the cardinality of a set. $O^{i,j}_t \in [0,1]$ and a value of 1 (0) represents a full agreement (disagreement). The spatial agreement over time is computed by averaging $O^{i,j}_t$ over Δt_3:

$$O^{i,j}_{\Delta t_3} = \frac{1}{\Delta t_3} \sum_{t}^{t+\Delta t_3} O^{i,j}_t. \tag{4.2}$$

In order to estimate the agreement for motion direction, we compute a score $r^{i,j}_{\Delta t_3}$ using the directional feature $\vec{\mathbf{d}}^k_t$ of each F^k [3] over Δt_3 that encodes the trajectory direction:

$$\vec{\mathbf{d}}^k_t = \left(u^k_{t+\Delta t_3} - u^k_t, v^k_{t+\Delta t_3} - v^k_t\right). \tag{4.3}$$

The directional similarity score $r^{i,j}_{\Delta t_3}$ is computed between F^i and F^j using the cosine similarity:

$$r^{i,j}_{\Delta t_3} = cos\left((\vec{\mathbf{d}}^i_t\ \vec{\mathbf{d}}^j_t)/(|\vec{\mathbf{d}}^i_t|\ |\vec{\mathbf{d}}^j_t|)\right), \tag{4.4}$$

where $r^{i,j}_{\Delta t_3} \in [-1,1]$ and negative values represent (estimated) targets moving in opposite directions.

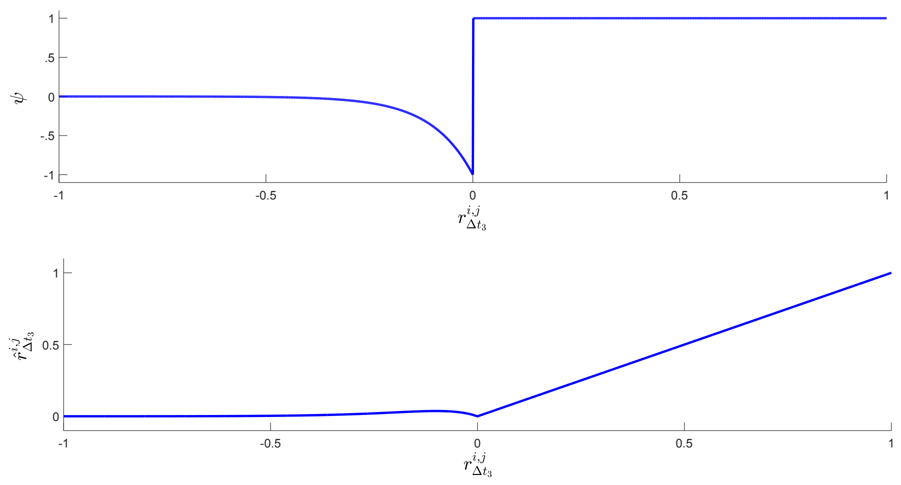


Figure 4.3: Weighting function ψ (top) and weighted directional score $\hat{r}^{i,j}_{\Delta t_3}$ (bottom) for directional feature normalization using λ = 10.

The primary objective is to determine the correlation score $R^{i,j}_{\Delta t_3}$ by combining $O^{i,j}_{\Delta t_3}$ and $r^{i,j}_{\Delta t_3}$. Since $O^{i,j}_{\Delta t_3} \in [0,1]$ and $r^{i,j}_{\Delta t_3} \in [-1,1]$, we aim to achieve a score $R^{i,j}_{\Delta t_3} \in [0,1]$. Furthermore, we are interested in an agreement on the direction of motion and are primarily interested in values $0 \leq r^{i,j}_{\Delta t_3} \leq 1$. Values of $r^{i,j}_{\Delta t_3}$ between the interval $-1 \leq r^{i,j}_{\Delta t_3} \leq 0$ represent trackers moving in different directions. Thus $r^{i,j}_{\Delta t_3}$ is normalized to [0,1], without ignoring values between the interval $-1 \leq r^{i,j}_{\Delta t_3} \leq 0$.

We therefore define a weighted directional similarity score $\hat{r}^{i,j}_{\Delta t_3}$ as:

$$\hat{r}^{i,j}_{\Delta t_3} = \psi(\lambda, r^{i,j}_{\Delta t_3})\, r^{i,j}_{\Delta t_3}, \tag{4.5}$$

where $\hat{r}^{i,j}_{\Delta t_3} \in [0,1]$ and $\psi \in [-1,1]$ is a weighting function that assigns a uniform weight to $0 \leq r^{i,j}_{\Delta t_3} \leq 1$, without ignoring values between the interval $-1 \leq r^{i,j}_{\Delta t_3} \leq 0$. Such a function is defined as:

$$\psi(\lambda, r^{i,j}_{\Delta t_3}) = \begin{cases} 1 & \text{if} \quad 0 \leq r^{i,j}_{\Delta t_3} \leq 1, \\ -e^{\lambda r^{i,j}_{\Delta t_3}} & \text{if} \quad -1 \leq r^{i,j}_{\Delta t_3} < 0, \end{cases} \tag{4.6}$$

where $\lambda \in (0,\infty)$ is the decay rate of ψ. Values of λ close to zero give smoother transitions for $\psi \in [-1,0]$ returning high values of $\hat{r}^{i,j}_{\Delta t_3}$ for $-1 \leq r^{i,j}_{\Delta t_3} \leq 0$. Whereas, high values of λ give abrupt transitions for $\psi \in [-1,0]$ returning values close to zero for $\hat{r}^{i,j}_{\Delta t_3}$ when $-1 \leq r^{i,j}_{\Delta t_3} \leq 0$. Figure 4.3 shows the relations between $r^{i,j}_{\Delta t_3}$ and $\hat{r}^{i,j}_{\Delta t_3}$ (bottom graph) and between $r^{i,j}_{\Delta t_3}$ and ψ (top graph) for $\lambda = 10$.

$R^{i,j}_{\Delta t_3}$ is finally computed as follows:

$$R^{i,j}_{\Delta t_3} = \omega\ O^{i,j}_{\Delta t_3} + (1-\omega)\ \hat{r}^{i,j}_{\Delta t_3}, \tag{4.7}$$

where $\omega \in [0,1]$ and is the weight employed to determine the influence of both $O^{i,j}_{\Delta t_3}$ and $\hat{r}^{i,j}_{\Delta t_3}$ in $R^{i,j}_{\Delta t_3}$. High (low) values of ω prioritize the spatial overlap (trajectory direction), which can be useful for short (long) Δt_3.

4.2.2 Partition generation

The spatio-temporal correlation scores between each pair of trackers, $R^{i,j}_{\Delta t_3}$ are then used to determine the partitions at each time step, $\mathbb{P}_{p,t}$, and hence the clusters (groups) within each $\mathbb{P}_{p,t}$. A single partition $\mathbb{P}_{p,t}$ of $\mathbf{F}$ is a collection of non-empty clusters $\mathbb{C}^a_{p,t}$, $a = 1, ..., |\mathbb{P}_{p,t}|$ such that each tracker in $\mathbf{F}$ is in exactly one $\mathbb{C}^a_{p,t}$. This implies that all $\mathbb{C}^a_{p,t}$ are mutually disjoint and the union of all $\mathbb{C}^a_{p,t}$ exhausts all of $\mathbf{F}$ (i.e. $\cup_{\mathbb{C}^a_{p,t} \in \mathbb{P}_{p,t}} \mathbb{C}^a_{p,t} = \mathbf{F}$) [53].

At each time step, K trackers can be grouped into multiple clusters $\mathbb{C}^a_{p,t}$, forming a single partition $\mathbb{P}_{p,t}$, where $|\mathbb{P}_{p,t}| \in [1,K]$. Let $[.]$ represent a partition. For example, the initial condition, $|\mathbb{P}_{p,t}| = 1$ implies that all trackers are in the same cluster i.e. $\left[\left\{F^1, ..., F^k\right\}\right]$. Whereas, $\left[\left\{F^1\right\}, ..., \left\{F^k\right\}\right]$ means that each tracker is a single cluster i.e. $|\mathbb{P}_{p,t}| = K$.

Our aim is to hypothesize a set of partitions $\{\mathbb{P}_{p,t}\}_{p=1}^{\mathscr{B}}$ to cluster the trackers. All possible partitions $\mathbb{P}_{p,t}$ can be systematically enumerated with an exhaustive search [86]. The set size is given by the Bell number $\mathscr{B}$ [33], which increases exponentially with K. For example, with $K = 8$ trackers $\mathscr{B} = 4140$ partitions are generated. An example of the generated partitions and clusters for two and three trackers is represented as a tree structure in Figure 4.4. The tree structure illustrates that two possible partitions are generated for two trackers. However, increasing the number of trackers to three increases the number of possible partitions to five.

To reduce the computational complexity, we use a greedy search that determines the most plausible partitions for a given number of clusters. Since the optimum number and composition of clusters is unknown, we take advantage of the hierarchical structure of the tracker relationships to generate a set of partitions whose cardinality is at most K.

We use hierarchical clustering (HC) [67] as the greedy search algorithm that has a linear relationship between the size of $\{\mathbb{P}_{p,t}\}_{p=1}^{K}$ and K, which significantly speeds up the search. Based on the pair-wise correlation scores $R^{i,j}_{\Delta t_3}$ between the trackers, a dendrogram is obtained which is

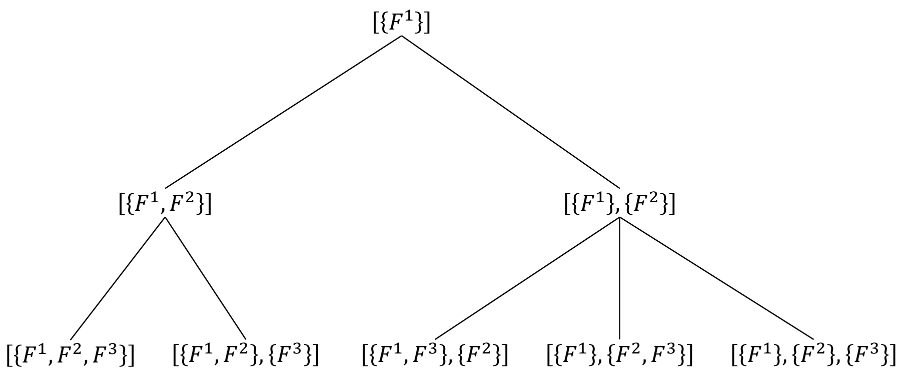


Figure 4.4: An illustration of increasing number of partitions with increasing trackers where [.] and {.} represents a partition and cluster, respectively. Starting from one tracker (F^1) on the top, the number of possible partitions are two for two trackers (F^1 and F^2), that increase to five with three trackers (F^1, F^2 and F^3).

inspected by a divisive (top-down) approach to determine each partition $\mathbb{P}_{p,t}$. The search starts with the partition that groups all trackers in one cluster $\mathbb{P}_{1,t}$. Recursively moving down the tree, a different $\mathbb{P}_{p,t}$ is generated at each level, with the final partition having each tracker in a separate cluster $\mathbb{P}_{K,t}$. A partition $\mathbb{P}_{p,t}$ is obtained as:

$$\mathbb{P}_{p,t} = H(\xi^*(p)), \tag{4.8}$$

where $p = 1,...,K$ and $H(\xi^*)$ is a HC-based function. $H(\xi^*(p))$ provides a partition for each p given the optimum distance threshold ξ^*, which is determined as:

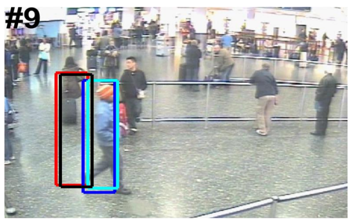
$$\xi^*(p) = \underset{\xi}{\operatorname{argmin}} \{J(\xi)\}, \tag{4.9}$$

where $\xi \in \{[0,(1-R^{i,j}_{\Delta t_3})] : i,j = 1,...,K\}$. Figure 4.5(a) and Figure 4.5(c) show an example for four trackers and the scores for their spatio-temporal relations, which are used to compute the dendrogram illustrated in Figure 4.5(b).

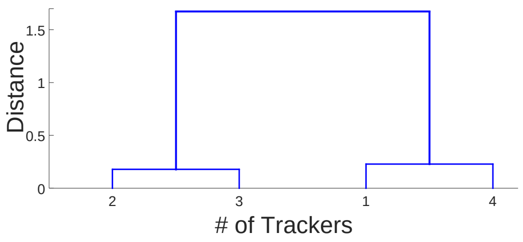
4.2.3 Partition validation

After generating the set of partitions $\{\mathbb{P}_{p,t}\}_{p=1}^{K}$, the objective is to select the optimal partition $\mathbb{P}^*_t$, i.e. the partition that best represents the spatio-temporal relations among trackers. We therefore define the score $S(\mathbb{P}_{p,t})$ as:

$$S(\mathbb{P}_{p,t}) = \frac{1}{|\mathbb{P}_{p,t}|} \sum_{a=1}^{|\mathbb{P}_{p,t}|} Q\left(\mathbb{C}^a_{p,t}\right), \tag{4.10}$$



(a)



(b)

Feature	Tracker pair (i,j)					
	F^1,F^2	F^1,F^3	F^1,F^4	F^2,F^3	F^2,F^4	F^3,F^4
$O^{i,j}_{\Delta t_3}$	.013	.00	.740	.766	.035	.017
$\hat{r}^{i,j}_{\Delta t_3}$	.029	.076	.984	.993	.135	.255
$R^{i,j}_{\Delta t_3}$	.021	.038	.862	.880	.085	.136

(c)

p	$\mathbb{P}_{p,t}$	$Q(\mathbb{C}^a_{p,t})$	$S(\mathbb{P}_{p,t})$
1	$[\{F^1\}\ \{F^2\}\ \{F^3\}\ \{F^4\}]$	[.165, .165, .165, .165]	.66
2	$[\{F^1\}\ \{F^2,F^3\}\ \{F^4\}]$	[.12, .88, .14]	.38
3	$[\{F^1,F^4\}\ \{F^2,F^3\}]$	[.88, .86]	.87
4	$[\{F^1,F^2,F^3,F^4\}]$	[.34]	.34

(d)

Figure 4.5: (a) Tracking results for frame 9 of the MCTTR0205a sequence (TRECVID) —: F^1; —: F^2; —: F^3; —: F^4. (b) The dendrogram obtained by hierarchical clustering. (c) Pair-wise tracker correlations scores. (d) Hypothesized partitions and cluster scores, where $\mathbb{P}_{3,t}$ has the highest score.

where $Q\left(\mathbb{C}^a_{p,t}\right)$ is the score for a single cluster $\mathbb{C}^a_{p,t} \in \mathbb{P}_{p,t}$. $S\left(\mathbb{P}_{p,t}\right)$ determines the partition $\mathbb{P}^*_t$ as:

$$\mathbb{P}^*_t = \underset{p}{\operatorname{argmax}} \left\{ S\left(\mathbb{P}_{p,t}\right) \right\}, \tag{4.11}$$

where $p = 1,...,K$ and $Q\left(\mathbb{C}^a_{p,t}\right)$ is dependent upon the pair-wise relationship score $R^{i,j}_{\Delta t_3}$ between trackers F^i and F^j in $\mathbb{C}^a_{p,t}$. The score $Q\left(\mathbb{C}^a_{p,t}\right)$ is computed as:

$$Q\left(\mathbb{C}^a_{p,t}\right) = \begin{cases} \frac{1}{v} \sum_{i=1}^{|\mathbb{C}^a_{p,t}|} \sum_{j=1}^{|\mathbb{C}^a_{p,t}|} R^{i,j}_{\Delta t_3} & \text{if} \quad \left|\mathbb{C}^a_{p,t}\right| > 1, \\ \\ 1 - \max_{b \in 1,...,|\mathbb{P}_{p,t}|} \left(Q\left(\mathbb{C}^a_{p,t} \bigcup \mathbb{C}^b_{p,t}\right)\right) & \text{if} \quad \left|\mathbb{C}^a_{p,t}\right| = 1, \end{cases} \tag{4.12}$$

where $v = \binom{\left|\mathbb{C}^a_{p,t}\right|}{2}$ is the total number of tracker-pair combinations within the cluster. Since a pair-wise score for a single tracker in a cluster, $\left|\mathbb{C}^a_{p,t}\right| = 1$, cannot be obtained, we compute its pair-wise scores with trackers in other clusters. Therefore $\mathbb{C}^a_{p,t} \bigcup \mathbb{C}^b_{p,t}$ indicates the hypothetical case where the tracker in $\mathbb{C}^a_{p,t}$ becomes part of cluster $\mathbb{C}^b_{p,t}$, and b is any of the remaining clusters within $\mathbb{P}_{p,t}$ $(b \neq a)$.

When each tracker is a single cluster, i.e. $|\mathbb{P}_{p,t}| = K$, each cluster score $Q\left(\mathbb{C}^a_{p,t}\right)$ is computed as:

$$Q\left(\mathbb{C}^a_{p,t}\right) = 1 - Q\left(\mathbb{C}^b_{p,t}\right), \tag{4.13}$$

where $\mathbb{C}^b_{p,t}$ is the cluster containing all trackers. Figure 4.5(d) shows the computed cluster and

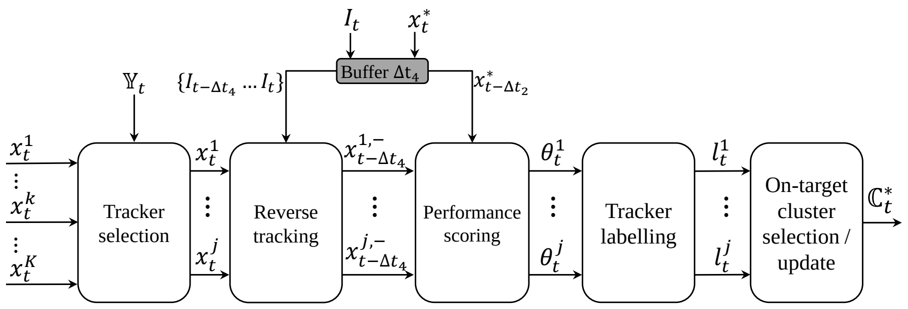


Figure 4.6: Block diagram of the reverse evaluation that identifies the *on-target* cluster $\mathbb{C}_t^*$.

partition scores, where $\mathbb{P}_{3,t}$ achieves the highest score.

4.2.4 Split-Merge detection

After determining $\mathbb{P}_t^*$, we employ an *online performance evaluation* method to identify the *on-target* cluster $\mathbb{C}_t^* \in \mathbb{P}_t^*$. We then propagate $\mathbb{C}_t^*$ over time if no changes occur within $\mathbb{C}_t^*$, where such changes indicate trackers leaving or joining $\mathbb{C}_t^*$. Thus, $\mathbb{C}_t^*$ cannot be propagated to the next time step and performance evaluation is required to identify trackers that may have lost the target or recovered from a failure. At each time step t, we identify these changes via the *split-merge detection* step by comparing the structure of $\mathbb{P}_t^*$ and the partition at the previous time step, $\mathbb{P}_{t-1}^*$. Then, we employ the online performance evaluation over a set of trackers $\mathbb{Y}_t$ that are selected as:

$$\mathbb{Y}_t = \begin{cases} \mathbb{C}_{t-1}^* & \text{if} \quad \mathbb{P}_{t-1}^* \equiv \mathbb{P}_t^*, \\ \mathbf{F} & \quad\;\; \text{otherwise}, \end{cases} \tag{4.14}$$

where the condition $\mathbb{P}_{t-1}^* \equiv \mathbb{P}_t^*$ checks the similarity between the number of clusters and their members (i.e. trackers). When this condition is satisfied, an existing cluster $\mathbb{C}_t^a \in \mathbb{P}_t^*$ equivalent to $\mathbb{C}_{t-1}^*$ is used at the current time step. However, in case of a split or merge, we evaluate the complete tracker set $\mathbf{F}$.

4.3 Adaptive time-reversed evaluation

We evaluate the performance of each tracker in the set $\mathbb{Y}_t$. This performance evaluation either determines the *on-target* cluster $\mathbb{C}_t^*$ from the partition $\mathbb{P}_t^*$ or validates the *on-target* cluster determined at the previous time step $\mathbb{C}_{t-1}^*$. We cast this problem as an online tracker performance evaluation and employ the time-reversed evaluation approach [149] over a sliding temporal window Δt_4.

Let *on-target* and *off-target* be the labels that indicate whether a tracker is following the target successfully or not, respectively. The goal of this performance evaluation is to identify the successful trackers given the outputs x_t^k and label them as:

$$x_t^k \rightarrow l_t^k \in \{on\text{-}target, off\text{-}target\}\,. \tag{4.15}$$

We assign l_t^k to trackers by identifying the cluster with the *on-target* trackers $\mathbb{C}_t^* = \{F^n\}_{n=1}^N \subseteq \mathbf{F}$ such that $N \leq K$. Trackers within $\mathbb{C}_t^*$ are assigned l_t^k = *on-target*, while the remaining trackers are labeled as *off-target*. In the following section, we first review the reverse-based evaluation method and then present our proposed improvements. A block diagram of the process is presented in Figure 4.6.

4.3.1 Time-reversed online evaluation

Reverse-based evaluation [149] measures the performance of a tracker during runtime based on the ability of the tracker to track the target in the time-reversed direction. For each frame where the tracker is evaluated, a reversed-tracker (i.e. the same tracker operating in reverse time) is used. Using the tracker output x_t^k, as the initialization for the reverse-tracker $x_t^{k,-}$, the output of the reverse-tracker is obtained as:

$$x_{t-1}^{k,-} = F^k(x_t^{k,-}, \mathbf{z}_t^k, \varsigma_t^k), \tag{4.16}$$

where $x_{t-1}^{k,-}$ is the reverse-tracker output at time $t-1$. Then the result of the reverse-tracker and that of the tracker are compared to obtain a similarity score θ_t^k (tracker performance score) by means of the Mahalanobis distance between the likelihood distributions of the forward and reverse target estimations. This comparison is performed at a certain time instant known as reference frame ($t = ref$) I_{ref}. $t = ref$ is the time instant where the tracker is known to be *on-target* and is set as $t = 1$ [149], i.e. the frame (I_1) where the target is initialized using GT.

This approach has two major limitations. First, the forward-reverse similarity uses the Mahalanobis distance that returns unbounded scores $\theta_t^k \in [0, +\infty)$, which can have different range of values depending on the trackers employed in the fusion framework. Hence θ_t^k may be inappropriate to compare the trackers to be combined. Second, running the reverse tracker until the first frame implies an exponential growth in computational time. A faster approximation (less computationally expensive) is proposed where I_{ref} is moved ahead in time based on fixed sized temporal windows. However, using fixed size temporal windows to update I_{ref} results in the reverse-tracker to accumulate errors over time, thus leading to drift [157]. For example, if the tracker loses the target and gets locked on the background the forward-reverse similarity may give high scores θ_t^k, since the reverse-tracker is incorrectly initialized by the wrong tracker estimations.

We address these shortcomings for reverse-tracking evaluation as described next.

4.3.2 Performance score and reference frame update

Reverse-tracking evaluates performance by comparing the outputs of the tracker and the reverse-tracker at I_{ref}. To employ this approach within a tracker fusion framework, we use the fused output determined at I_{ref} (x^*_{ref}) as a common point of evaluation for all trackers.

First, we address the limitation of unbounded θ_t^k scores by comparing the results of the reverse-tracker with the fused output to obtain a $\theta_t^k \in [0,1]$ as:

$$\theta_t^k = G(x_{ref}^{k,-}, x^*_{ref}), \tag{4.17}$$

where $x_{ref}^{k,-}$ and x^*_{ref} are the reverse-tracker and fused outputs at I_{ref}, respectively; G defines the output similarity and is computed using Eq. 4.1 where A_t^i and A_t^j are replaced by $A_{ref}^{x^*}$ and $A_{ref}^{k,-}$, respectively. $A_{ref}^{x^*}$ and $A_{ref}^{k,-}$ are the sets containing the pixels of the bounding boxes of x^*_{ref} and $x_{ref}^{k,-}$, respectively.

Then, for the limitation of the exponential growth of computational time when I_{ref}=1, we update I_{ref} over time so that the computational cost is reduced and reverse-evaluation can be applied to long sequences. We implement such update assuming that the fused output is *on-target* and that the target has changed position from I_{ref} to the current frame I_t, thus making the motion information useful for reverse-analysis.

The motion of bounding boxes is minimal when the tracker is *on-target* and the target is static; or when the tracker drifts from the target and gets locked onto a static background region. Because it is difficult to differentiate between these two situations, we analyze significant motion changes of the trackers compared with their average motion. The maximum motion M^k is computed over temporal window Δt_4 using F^k trajectory, and the top-left $(u_{1,t}^k, v_{1,t}^k)$ and bottom-right $(u_{2,t}^k, v_{2,t}^k)$ coordinates of the bounding box. Motion for u_1^k over Δt_4 is computed as:

$$M_{u_1^k} = \frac{1}{\Delta t_4} \sum_{t'=t-\Delta t_4}^{t} \left(M_{u_{1,t'}^k} - M_{u_{1,t'-1}^k} \right). \tag{4.18}$$

Motion for $v_{1,t}^k, u_{2,t}^k$ and $v_{2,t}^k$ is computed using Eq. 4.18, where $M_{u_1^k}$ is replaced by $M_{v_1^k}$, $M_{u_2^k}$ and $M_{v_2^k}$, respectively. $M^k = \max(M_{u_1^k}, M_{v_1^k}, M_{u_2^k}, M_{v_2^k})$ returns the maximum motion for F^k. The motion of the fused output M^{x^*} is determined as:

$$M^{x^*} = \frac{1}{\Delta t_4} \sum_{t'=1}^{t} (M_{x_{t'}^*} - M_{x_{t'-1}^*}), \tag{4.19}$$

and is used as a common threshold to compare the motion of all trackers in the framework.

The performance of each tracker is computed using Eq. 4.17. To determine a single I_{ref} for all trackers, we use $\max(M^k)$ and $\max(\theta_t^k)$ to select the best performing tracker for that temporal window. Using both motion analysis and performance of the tracker, we adaptively estimate and update I_{ref}. The value of I_{ref} is updated using the temporal window Δt_4 when tracker performance ($\max(\theta_t^k)$) and motion ($\max(M^k)$) are above a threshold τ_2 and M^{x^*}, respectively. I_{ref} remains unchanged otherwise and is determined as:

$$I_{ref} = \begin{cases} I_{t-\Delta t_4} & \text{if} \max(M^k) \geq M^{x^*} \text{and} \max(\theta_t^k) \geq \tau_2, \\ I_{ref} & \text{otherwise}, \end{cases} \tag{4.20}$$

where $\tau_2 = 0.5$ is the minimum tracker accuracy [125].

4.3.3 Selection or update of the *on-target* cluster

Reverse-evaluation identifies the *on-target* trackers by using the individual performance scores θ_t^k of trackers in $\mathbb{Y}_t$. Trackers with $\theta_t^k \geq \tau_2$ are labeled as *on-target*, enabling the method to select $\mathbb{C}_t^*$ as the cluster $\mathbb{C}_t^a$ with all *on-target* trackers:

$$\mathbb{C}_t^* = \{\mathbb{C}_t^a \in \mathbb{P}_t^* : l_t^k = \textit{on-target}, \forall F_t^k \in \mathbb{C}_t^a\}. \tag{4.21}$$

$\mathbb{C}_t^*$ is propagated until the detection of a split or a merge (Section 4.2.4), which happens when trackers leave or join the cluster $\mathbb{C}_t^*$, respectively. A split or merge indicates that some or all of the *on-target* trackers may have failed. When such changes occur, all trackers are re-evaluated to determine the new *on-target* cluster $\mathbb{C}_t^*$. One advantage of the propagating $\mathbb{C}_t^*$ over time is that it helps to reduce the computational load by avoiding to apply reverse-evaluation over all trackers when they maintain their spatio-temporal relationships over time.

4.4 Tracker-level fusion

To determine the final target state (fused output), we combine the estimated outputs of the trackers belonging to the *on-target* cluster $\mathbb{C}_t^*$. We employ both an equal weighted and a performance-based weighted average of outputs approach. The tracking performance score y_t^n, that is based on the ability of the tracker to distinguish the target from the background (Section 3.2) is used as the tracker weight. The equal weighted approach estimates the final target state x_t^* as:

$$x_t^* = \frac{1}{N}\sum_{n=1}^{N} x_t^n. \tag{4.22}$$

Using y_t^n and the trackers belonging to $\mathbb{C}_t^*$, the final target state based on the weighted approach x_t^{**} is determined as:

$$x_t^{**} = \frac{1}{N}\sum_{n=1}^{N} \check{y}_t^n \, x_t^n, \tag{4.23}$$

where $\check{y}_t^n$ is the normalized weight for each tracker n and is determined as:

$$\check{y}_t^n = \frac{y_t^n}{\sum_{n=1}^{N} y_t^n}. \tag{4.24}$$

4.5 Experimental setup

We evaluate the accuracy of the Tracker Partition Fusion (TPF) framework both in terms of tracking accuracy and the accuracy in assigning the *on-target* and *off-target* labels to trackers and clusters. We test the TPF framework over the dataset D1 (Appendix A) using eight trackers: ST[1], AFT[2], LOT[3], IVT[4], FCT[5], AMS[6], L1T[7] and LSST[8] (Section 2.2.5) and use the publicly available implementations of the authors.

The fusion module is tested using both the equal weighted and the performance-based weighted

[1]http://faculty.ucmerced.edu/mhyang/project/tip13_prototype/TIP12-SP.htm
[2]http://web.cs.hacettepe.edu.tr/~erkut/#publications
[3]http://www.eng.tau.ac.il/~oron/LOT/LOT.html
[4]http://www.cs.toronto.edu/~dross/ivt/
[5]http://www4.comp.polyu.edu.hk/~cslzhang/FCT/FCT.htm
[6]http://www4.comp.polyu.edu.hk/~cslzhang/SOAMST.htm
[7]http://www.dabi.temple.edu/~hbling/code_data.htm
[8]http://202.118.75.4/lu/publications.html

Table 4.1: Combinations of trackers used for each of the proposed approaches.

Proposed approach		**Selected Trackers**	
Non-weighted	Weighted	# of Trackers	Combination
TPF_3^*	TPF_3^{**}	3	ST, AFT, LOT
TPF_4^*	TPF_4^{**}	4	ST, AFT, LOT, IVT
TPF_5^*	TPF_5^{**}	5	ST, AFT, LOT, IVT, FCT
TPF_6^*	TPF_6^{**}	6	ST, AFT, LOT, IVT, FCT, AMS
TPF_7^*	TPF_7^{**}	7	ST, AFT, LOT, IVT, FCT, AMS, L1T
TPF_8^*	TPF_8^{**}	8	ST, AFT, LOT, IVT, FCT, AMS, L1T, LSST

approach, and apply six different combinations of the eight selected trackers (Section 2.2.5). The combinations and symbols used for both approaches are listed in Table 4.1. We use TPF_3^* only for Section 4.6 and 4.7, while Sections 4.8 and 4.9 use all six configurations of each of the two approaches.

This section presents the experimental setup required for evaluating the complete framework. First, we present the details of the experimental parameters used within the framework in Section 4.5.1, which is followed by details of the evaluation measures in Section 4.5.2 and details of the SOA methods in Section 4.5.3.

4.5.1 Experimental parameters

TPF uses two temporal windows Δt_3 and Δt_4 to buffer data from the future and past time instants, respectively. $\Delta t_4 = 10$ employed for the reverse-analysis provides a good speed-accuracy trade-off as shown in [149] and its value is updated if the motion or the performance of trackers is below the thresholds (Section 4.3.2). The second temporal window used for tracklet correlation is set to $\Delta t_3 = 10$ to keep an initial forward-backward symmetry for analysis, since no prior information is available to define the importance of one over the other. To determine the influence of both the spatial and temporal features Eq. 4.7 on the clustering approach, we test and compare results for $\omega = 0, 0.5$ and 1, where $\omega = 0$ (1) assigns complete weight to the spatial (temporal) feature. $\omega = 0.5$ ensures equal weighting of both the spatial and temporal features. For Eq. 4.5, we heuristically found that $\lambda \in [5, 15]$ gives the desired ψ behaviour i.e. assign uniform weights to $0 \leq r_{\Delta t_3}^{i,j} \leq 1$ and lower weights to $-1 \leq r_{\Delta t_3}^{i,j} \leq 0$ (Section 4.2.1). We therefore use the mean value i.e. $\lambda = 10$. Finally, the thresholds for determining tracking performance online (τ_2) and offline (τ_3) are set to 0.5 as done previously [125].

To implement the hierarchical clustering approach (Section 4.2.2), we use the built-in MATLAB *cluster* and *linkage* functions from the *Statistics and Machine learning* toolbox for Eq. 4.8

and Eq. 4.9, respectively. The *linkage* function is based on the *nearest neighbor* clustering approach.

4.5.2 Evaluation measures

We measure the performance of TPF in assigning the *on-target* and *off-target* labels to trackers and clusters from the valid partition $\mathbb{P}_t^*$. We employ the GT information to compute the overlap score for each cluster $O_t^{\mathbb{C}^a}$ by taking the average of $(1-O_t)$ (Eq. 3.11) for the trackers within the cluster. The *on-target* trackers are defined for $(1-O_t) \geq \tau_3$ and $O_t^{\mathbb{C}^a} \geq \tau_3$ defines the *on-target* cluster $\mathbb{C}_t^*$. We use n_{TP}, n_{FP}, n_{FN} and n_{TN} as defined in Section 3.4.2 to measure the performance of TPF in assigning the *on-target* and *off-target* labels. For TPF, n_{TP} (n_{FP}) and n_{TN} (n_{FN}) are the number of clusters or trackers correctly (incorrectly) labelled as *on-target* and *off-target*, respectively. Using n_{TP}, n_{FP}, n_{TN} and n_{FN} we then compute the precision, P, the recall, R and the F-score as defined in Section 3.4.2.

To compare TPF with the selected trackers and the SOA tracking methods, we measure the tracking performance as mean overlap score (O_G) over the complete sequence as:

$$O_G = \frac{1}{T}\sum_{t=1}^{T}(1-O_t), \tag{4.25}$$

where O_t is measured as the overlap score between the tracker output and the GT data (Eq. 3.11).

4.5.3 Selected benchmark methods

We compare the tracking performance of TPF with the eight selected trackers, two recent trackers: STRUCK (STR) [57, 58] and Kernelized Correlation Filters (KCF) [62]; and three SOA tracker-level fusion approaches: Average fusion (AvgF), Symbiotic Tracker (SymT) [49] and Visual Tracker Sampler (VTS) [74]. STRUCK [57, 58] is a tracking-by-detection approach that employs SVMs with Gaussian kernels. We test the three features used by the method (Haar, raw pixels and histograms) and report the results for histograms as they outperform Haar features and raw pixels. KCF [62] employs correlation among filters based on histograms of oriented gradients features. AvgF combines the eight trackers by assigning equal weights to each tracker i.e. without employing any performance evaluation. SymT estimates trackers' relationships based on their spatial agreement only, and individual tracker performance is based on displacements between consecutive frames. SymT however does not employ any mechanism to determine if

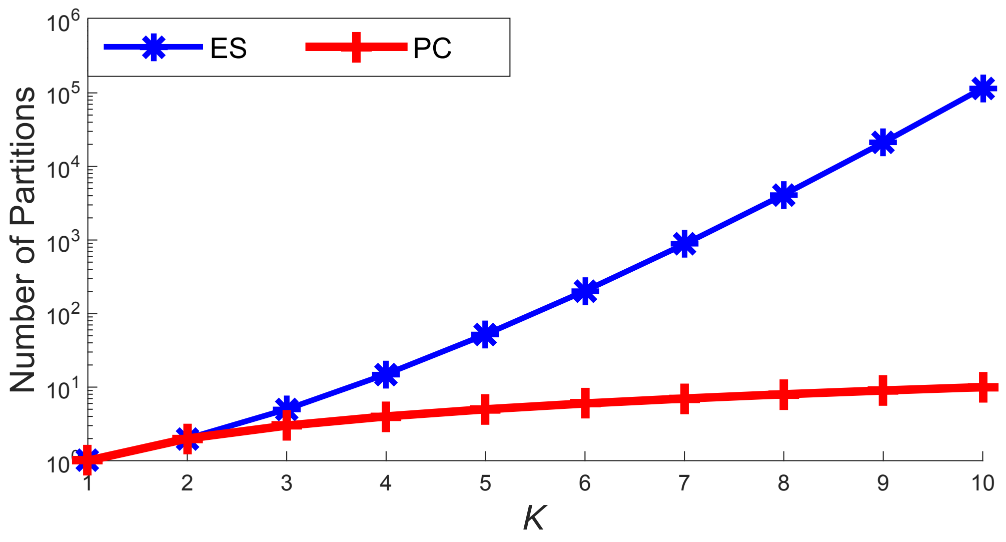


Figure 4.7: Comparison of methods to generate partitions $\{\mathbb{P}_{p,t}\}_{p=1}^{B}$ of K trackers. With exhaustive search (ES) B grows exponentially, while the proposed clustering (PC) is bounded by $B = K$.

the trackers are on the right target and fuses the outputs of all trackers. We re-implement SymT as described in [49] and generated results using the eight selected trackers. VTS combines two motion and four appearance models to get eight trackers and then uses a likelihood-based tracker performance score to select the output of the best tracker as the final output. STR[9], KCF[10] and VTS[11] are tested by using publicly available authors' implementations.

4.6 Experimental analysis of tracker clustering

This section first presents a comparison of the proposed hierarchical clustering approach to the exhaustive search approach for determining partitions of the trackers. Next, we compare our approach of combining both features (spatial and temporal) to using each feature individually (either spatial or temporal) for determining the pair-wise tracker relationships.

4.6.1 Comparison of the proposed clustering with exhaustive search

Figure 4.7 compares the generated set of partitions $\{\mathbb{P}_{p,t}\}_{p=1}^{B}$ with an increasing number of trackers K for both approaches. The accuracy of the tracking results is equal as the proposed clustering (PC) and exhasutive search (ES) select the same valid partition. However, with ES the size grows exponentially with an increasing K whereas PC keeps the size of $\{\mathbb{P}_{p,t}\}_{p=1}^{B}$ bounded with

[9] http://www.samhare.net/research/struck
[10] http://www.robots.ox.ac.uk/~joao/circulant/index.html
[11] http://cv.snu.ac.kr/research/vtdvts/

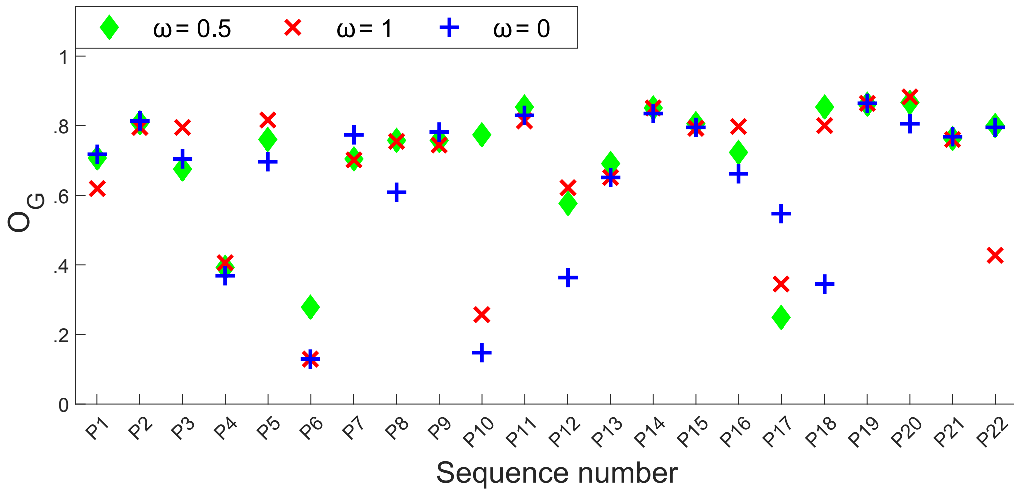


Figure 4.8: Tracker accuracy O_G using individual features, (ω = 1 for overlap and ω = 0 for direction) and their equal combination (ω = 0.5) as used in Eq. 4.7.

respect to $B = K$.

4.6.2 Performance analysis of features

TPF combines the features $O^{i,j}_{\Delta t_3}$ and $\hat{r}^{i,j}_{\Delta t_3}$ to determine the pair-wise tracker correlation scores $R^{i,j}_{\Delta t_3}$. Using either of the two features individually directly impacts $R^{i,j}_{\Delta t_3}$, which can lead to incorrect estimation of the best partition $\mathbb{P}^*_t$ and hence the *on-target* cluster $\mathbb{C}^*_t$. For e.g. using only $\hat{r}^{i,j}_{\Delta t_3}$ (i.e. $\omega = 0$ in Eq. 4.7) leads to high $R^{i,j}_{\Delta t_3}$ scores for a pair of trackers that move in the same direction but are spatially far from each other. Similarly being dependent only on the similarity of spatial location gives higher importance to tracker pairs with high values of $O^{i,j}_{\Delta t_3}$. Since no feature is optimum for all situations, combining both features increases the overall performance in comparison to using them individually. A comparison based on the tracking accuracy O_G of the three features is presented in Figure 4.8. Using ω = 0.5 allows the method to correctly estimate $\mathbb{P}^*_t$ and $\mathbb{C}^*_t$, hence improving the results globally in 60% of the sequences.

Based on the F-score, we present an accuracy comparison at both cluster and tracker-level for assigning *on-target* and *off-target* labels in Table 4.2. At both tracker and cluster-level TPF improves or has similar performance in comparison to using either of the features individually in 60% of the sequences. Furthermore, at tracker-level the F-score improves by 5% (7%) when using both features in comparison to using only the overlap (direction) feature. Similarly at cluster-level an improvement of 6% (7%) is observed in comparison to the overlap (direction) feature. The overall results indicate that on average the accuracy of combining both features

Table 4.2: Comparison of feature combinations for the proposed approach. Results show the F-score at tracker-level and cluster-level, with different feature weights ω in Eq. 4.7.

	Tracker-level			Cluster-level		
	ω=0	ω=0.5	ω=1	ω=0	ω=0.5	ω=1
P1	.92	.90	.87	.94	.92	.89
P2	.98	.98	.97	.99	.99	.98
P3	.91	.92	.97	.87	.90	.98
P4	.87	.90	.94	.80	.87	.92
P5	.87	.90	.94	.93	.93	.99
P6	.44	.48	.45	.38	.42	.40
P7	.86	.78	.84	.99	.87	.87
P8	.88	.94	.93	.90	.97	.96
P9	.87	.81	.82	.99	.95	.84
P10	.46	.82	.37	.57	.90	.34
P11	.92	1	.94	1	1	.98
P12	.63	.77	.83	.60	.75	.83
P13	.88	.95	.87	.87	.98	.90
P14	.95	.96	.98	1	1	1
P15	.96	.97	.98	1	1	.99
P16	.81	.81	.93	.81	.88	1
P17	.52	.55	.56	.53	.67	.47
P18	.51	.93	.88	.60	.99	.96
P19	.98	.98	.98	1	1	1
P20	.95	1	.98	.97	1	1
P21	.95	.96	.96	.95	.98	.99
P22	.91	.95	.52	.99	1	.60
Mean	.82	**.88**	.84	.85	**.91**	.86

increases in comparison to using single features.

We further extend the experimental analysis to determine the statistical significance of the results achieved by using ω = 0.5 as opposed to ω = 0 or 1. We use the two sample Student's t-test assuming unequal variances to perform a significance test. We define processes X, Y1 and Y2 as follows:

$$\begin{aligned}
&\text{X}: &&\omega = 0.5, &&\mu_X = 0.88 \text{ (tracker-level) or } 0.91 \text{ (cluster-level)},\\
&\text{Y1}: &&\omega = 0, &&\mu_{Y1} = 0.82 \text{ (tracker-level) or } 0.85 \text{ (cluster-level)},\\
&\text{Y2}: &&\omega = 1, &&\mu_{Y2} = 0.84 \text{ (tracker-level) or } 0.86 \text{ (cluster-level)}.
\end{aligned}$$

Using results obtained from Y1 and Y2 as base-line, a standard significance value of 0.10

and the total number of sequences (twenty two) as the sample, we perform the significance test on the results obtained from process X.

The first test that compares X with Y1 returns a p-value of 0.25 (more than the significance value) and fails to reject the null hypothesis H_0 under the following hypothesis:

$$\begin{aligned} H_0: &\quad \mu_X = \mu_{Y1}, \\ H_1: &\quad \mu_X \neq \mu_{Y1}. \end{aligned} \tag{4.26}$$

where H_0 is rejected (H_1 is accepted) if X performs better than Y1.

A similar test compares X with Y2 using the hypothesis:

$$\begin{aligned} H_0: &\quad \mu_X = \mu_{Y2}, \\ H_1: &\quad \mu_X \neq \mu_{Y2}, \end{aligned} \tag{4.27}$$

where H_0 is accepted with a p-value of 0.49.

Furthermore, we also compare the two extremes of the data i.e. Y1 and Y2 using the same test under the following hypothesis:

$$\begin{aligned} H_0: &\quad \mu_{Y2} = \mu_{Y1}, \\ H_1: &\quad \mu_{Y2} \neq \mu_{Y1}, \end{aligned} \tag{4.28}$$

where H_0 is accepted with a p-value of 0.69.

We perform similar tests for the results obtained from ”cluster-level” that return similar test results. When comparing with Y1, a p-value = 0.25 accepts H_0 in favour of Y1, while a p-value of 0.36 accepts H_0 in favour of Y2. Comparing Y1 and Y2 returns a p-value of 0.87, hence accepting H_0. From the following results it can be implied that experiments on additional video sequences can generate comparable results for X (ω = 0.5) in comparison to Y1 (ω = 0) and Y2 (ω = 1). However, the smaller p-value for Y1 in comparison to Y2 casts doubt on the validity of H_0. Furthermore, the comparison between Y1 and Y2 reveals that neither of the two extreme values (ω = 0, 1) are the best values as they might generate similar results. The inference based on this significance test results indicates that the optimal value for ω can be found by performing additional experiments and varying the value between $0.5 < \omega < 1$.

Table 4.3: F-score with (TPF) and without (TPF') motion analysis for the three fused trackers with the reference frame I_{ref} updated using motion analysis (Section 4.3). Key - ST: Sparse Tracker, AFT: Adaptive Fragments based Tracker; LOT: Locally Orderless Tracker.

	ST [143]		AFT [42]		LOT [104]		Sequence mean for all trackers	
	TPF'	TPF	TPF'	TPF	TPF'	TPF	TPF'	TPF
P1	.92	.88	.96	.92	.94	.90	.94	.90
P2	1	1	1	.97	1	.97	1	.98
P3	.95	.99	.95	.91	1	.87	.97	.92
P4	.85	.94	.78	.95	.59	.74	.74	.88
P5	.81	.87	.94	.92	.61	.76	.79	.85
P6	.26	.41	.41	.60	.52	.43	.40	.48
P7	.53	.67	.74	.82	.63	.78	.63	.76
P8	.88	.87	.99	.94	1	.98	.95	.93
P9	.90	.72	.90	.78	.89	.91	.89	.80
P10	.88	.75	.91	.83	.88	.82	.89	.80
P11	1	1	1	1	1	1	1	1
P12	.57	.82	.61	.66	.46	.85	.55	.77
P13	.79	.98	.83	.96	.94	.92	.86	.95
P14	1	.98	.96	.95	.91	.94	.96	.96
P15	.99	.98	.96	.96	.95	.99	.97	.98
P16	.91	.85	.99	.86	.88	.73	.93	.81
P17	.51	.54	.80	.40	.88	.48	.73	.47
P18	.29	.33	.05	.87	.08	1	.14	.73
P19	1	1	1	1	1	.95	1	.98
P20	1	1	.97	.99	1	1	.99	1
P21	.94	.88	1	1	1	.99	.98	.96
P22	.92	.96	.81	.97	.92	.86	.88	.93
Tracker mean	.81	**.83**	.84	**.87**	.82	**.86**	.83	**.85**

4.7 Experimental analysis of the adaptive time-reversed evaluation

This section presents an experimental evaluation of the definition of the adaptive reference frame for online performance analysis (Section 4.3.2). We first compare our approach of employing motion analysis for determining the adaptive reference frame to the case of not using motion analysis. Then, we compare our approach of adaptively updating the reference frame with the original approach where the reference frame is updated with out any prior knowledge [149].

4.7.1 Performance analysis for motion

Table 4.3 compares the proposed approach with and without motion analysis (TPF and TPF', respectively) to update I_{ref}, in terms of the F-score for selecting the *on-target* trackers. Results for P4-P7 indicate the case when trackers might lose the target due to background clutter and get fixed on the background. Since TPF' is unable to detect this situation in P4, it determines the trackers to be always *on-target*. TPF improves TPF' by 10%, 22% and 25% for ST, AFT and LOT, respectively (19% mean improvement). For P5, P6 and P7 TPF improves performance by 8%, 20% and 20%, respectively. P12 remains occluded between frames 29-39 where ST loses the target and becomes locked on to foreground objects being labeled as *on-target* by TPF', whereas AFT and LOT are labeled as off-target. Using motion, TPF improves by 44%, 8% and 85% (40% mean improvement). For P18, ST loses the target at frame 7 due to similar background. TPF' assumes ST to be *on-target*, while AFT and LOT are labeled as *off-target*. TPF uses motion to correctly label AFT and LOT as *on-target* achieving an overall improvement of 420% in comparison to TPF'. ST remains *on-target* for the first 6 frames of the sequence; where TPF incorrectly labels it *off-target* in 5 out of the 6 frames, hence resulting in lower values for TPF. For P17, the target does not move for most of the sequence. ST and AFT lose the target at frame 45 due to similar background, and form a cluster. Due to the stationary target, TPF assumes the ST-AFT cluster to be *on-target* resulting in incorrect labels for all trackers, hence decreasing performance by 35%. Globally, TPF improves TPF' by 2%, 4% and 5% for ST, FT and LOT, respectively.

4.7.2 Performance analysis of the I_{ref} update

As opposed to using fixed-sized temporal windows as done in the original approach (OA) [149], TPF employs both motion analysis and performance evaluation of trackers to adaptively update I_{ref}. This allows TPF to minimize tracker drift, hence improving its performance in the identification of the *on-target* trackers. Figure 4.9 compares the proposed update for the reference frame I_{ref} with the OA. The average result for three trackers (ST, AFT, LOT) is presented in terms of the overlap score O_G between the GT and the existing forward estimation in I_{ref} obtained by OA and TPF. TPF improves OA in 16 out of 22 sequences. I_{ref} is updated only when tracker(s) are found to be *on-target*. For instance, all three trackers fail between frames 95-110 for P4. TPF is able to accurately detect this scenario and does not update I_{ref} after frame 110, whereas OA keeps moving I_{ref} forward in time, thus accumulating tracker errors. For P9 and P21, OA achieves higher accuracy since all of the three trackers are able to track the target throughout the sequences. For P15 and P16 TPF achieves similar tracking accuracy to OA, where the camera moves with the target allowing all the trackers to remain *on-target* for most part of the sequence.

Table 4.4: Mean overlap score comparison in terms of O_G (ten independent runs). Key - ST: Sparse Tracker [143], AFT: Adaptive Fragments based Tracker [42]; LOT: Locally Orderless Tracker [104]; IVT: Incremental Visual Tracker [110]; FCT: Fast Compressive Tracker [166]; AMS: Mean Shift Tracker [102]; L1T: L1 Tracker [93]; LSST: Least Soft-threshold Squares Tracker [142]; AvgF: Average Fusion; SymT: Symbiotic Tracker [49]; VTS: Visual Tracker Sampler [74]; STR: STRUCK [57, 58]; KCF: Kernelized Correlation Filter Tracker [62].

	Fused trackers								Proposed approaches												Selected state-of-the-art				
									Non-weighted fusion						Weighted fusion										
	ST	AFT	LOT	IVT	FCT	AMS	L1T	LSST	TPF_3^*	TPF_4^*	TPF_5^*	TPF_6^*	TPF_7^*	TPF_8^*	TPF_3^{**}	TPF_4^{**}	TPF_5^{**}	TPF_6^{**}	TPF_7^{**}	TPF_8^{**}	AvgF	SymT	VTS	STR	KCF
P1	.65	.70	.64	.64	.14	.72	.20	.22	.72	.70	.53	.67	.67	.50	.72	.72	.56	.69	.65	.50	.25	.35	.24	.75	.21
P2	.80	.76	.79	.82	.58	.23	.82	.50	.81	.80	.78	.74	.74	.73	.82	.83	.78	.77	.70	.74	.40	.65	.77	.68	.87
P3	.40	.70	.73	.57	.40	.47	.79	.75	.70	.68	.74	.70	.82	.82	.71	.66	.73	.74	.80	.80	.78	.81	.77	.78	.85
P4	.35	.43	.25	.26	.41	.30	.23	.31	.41	.41	.41	.42	.42	.42	.42	.38	.38	.44	.44	.44	.30	.33	.37	.39	.64
P5	.81	.82	.31	.64	.39	.39	.38	.75	.76	.71	.51	.70	.47	.57	.77	.70	.47	.67	.45	.55	.64	.72	.59	.68	.85
P6	.12	.14	.10	.13	.10	.29	.31	.10	.13	.12	.12	.12	.13	.12	.13	.14	.13	.11	.13	.11	.12	.12	.14	.13	.10
P7	.91	.79	.78	.91	.81	.81	.92	.93	.85	.86	.81	.85	.85	.89	.85	.88	.83	.87	.87	.87	.89	.89	.85	.80	.83
P8	.63	.72	.78	.43	.04	.60	.55	.41	.77	.77	.71	.65	.63	.58	.76	.78	.75	.64	.63	.56	.47	.62	.78	.74	.67
P9	.75	.74	.89	.81	.74	.75	.89	.83	.77	.83	.81	.80	.81	.82	.78	.84	.82	.81	.81	.83	.84	.84	.92	.82	.74
P10	.11	.15	.74	.11	.44	.70	.12	.12	.77	.75	.55	.48	.37	.76	.77	.74	.54	.48	.39	.74	.12	.12	.14	.91	.11
P11	.75	.74	.91	.81	.75	.12	.90	.92	.86	.85	.83	.77	.80	.82	.87	.85	.84	.78	.81	.83	.69	.90	.84	.54	.76
P12	.36	.52	.25	.35	.63	.33	.25	.35	.55	.40	.39	.40	.37	.35	.54	.38	.35	.40	.37	.34	.30	.33	.36	.28	.27
P13	.60	.63	.42	.55	.67	.48	.51	.52	.71	.63	.70	.68	.61	.67	.72	.60	.69	.71	.62	.66	.53	.55	.56	.69	.44
P14	.87	.68	.79	.85	.78	.73	.78	.87	.86	.83	.82	.76	.82	.76	.84	.81	.82	.74	.80	.74	.82	.83	.85	.82	.82
P15	.74	.80	.78	.74	.77	.61	.90	.89	.81	.76	.79	.78	.81	.82	.80	.79	.77	.78	.81	.82	.82	.83	.78	.69	.79
P16	.78	.79	.74	.56	.78	.80	.71	.92	.74	.85	.83	.82	.82	.89	.73	.86	.85	.81	.82	.88	.70	.77	.86	.78	.79
P17	.26	.22	.65	.37	.20	.31	.53	.28	.26	.37	.31	.30	.40	.36	.27	.41	.35	.30	.39	.36	.43	.41	.24	.16	.84
P18	.42	.33	.87	.10	.85	.79	.69	.86	.86	.89	.87	.81	.80	.85	.87	.88	.85	.77	.78	.85	.28	.38	.87	.84	.12
P19	.86	.81	.77	.87	.87	.56	.78	.86	.89	.87	.87	.90	.89	.89	.87	.88	.83	.90	.89	.89	.87	.88	.80	.76	.89
P20	.90	.79	.85	.89	.89	.58	.84	.79	.88	.85	.86	.85	.86	.85	.85	.85	.82	.85	.86	.85	.83	.85	.87	.78	.90
P21	.75	.73	.84	.67	.73	.78	.83	.83	.77	.79	.77	.73	.84	.74	.80	.76	.80	.74	.84	.75	.78	.79	.85	.75	.74
P22	.52	.42	.36	.52	.68	.80	.81	.13	.43	.54	.50	.51	.52	.50	.44	.52	.53	.51	.51	.48	.43	.54	.37	.13	.13
Mean	.57	.61	.65	.57	.58	.55	.62	.60	.70	.69	.66	.66	.66	.67	.70	.69	.66	.66	.65	.66	.56	.61	.62	.63	.61

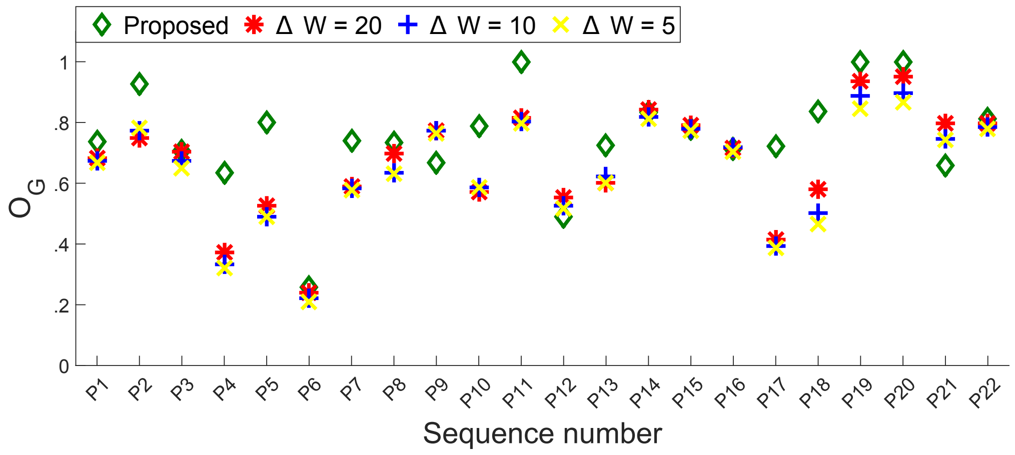


Figure 4.9: Comparison of I_{ref} selected by the proposed approach and the original approach based on fixed temporal windows $\Delta W = 5, 10, 20$.

4.8 Experimental analysis of tracking fusion framework

This section presents an experimental comparison of the TPF framework in terms of the tracking accuracy measured as the mean overlap score O_G (Eq. 4.25) over the complete sequence. We first present a discussion on the comparison of the TPF framework with the selected trackers, which is then followed by a discussion on the experimental results of the two fusion approaches (equal weighted and weighted). Finally, we discuss performance comparison of the TPF framework with the selected benchmark fusion and tracking methods. Table 4.4 presents results of the selected (fused) trackers, equal weighted fusion: $TPF_3^*....TPF_8^*$, weighted fusion: $TPF_3^{**}....TPF_8^{**}$ and the selected benchmark methods.

4.8.1 Combining trackers

A comparison of the six TPF configurations ($TPF_3^*, ..., TPF_8^*$) in Table 4.4 shows that TPF_3^* is the best and average tracking accuracy decreases with increasing number of trackers. There are two main reasons for this accuracy drop. First, low performing trackers in the *on-target* cluster decrease the overall tracking accuracy when fused using the average approach. This is indicated by results for P7, P9 and P15, where all trackers are *on-target* for most of the sequence. This can be further validated by a comparison of results with AvgF (Table 4.4) for these sequences. Second, the number of splitting and merging of clusters increases as we include more trackers, thus increasing the chances of wrong reverse-analysis evaluations. For P1, the target undergoes occlusions between frames 75-95. FCT, L1T and LSST lose the target due to occlusion, however FCT (a deterministic tracker) achieves the best performance score during this interval, reducing the overall accuracy. Results for TPF_5^* and TPF_8^* indicate this scenario. Similarly for P10, a drop in accuracy of TPF_5^*, TPF_6^* and TPF_7^* occurs when the target undergoes occlusions between frames 20-35. All trackers lose the target at frame 20. However, LOT and AMS regain the target. The target remains stationary from frame 45 till the end of the sequence. This scenario allows failed trackers to achieve a higher performance score during reverse-analysis, hence reducing the tracking accuracy. TPF removes low-performing trackers to improve the overall tracking accuracy. Results for P3, P10, P13, P18 and P19 indicate that TPF outperforms all trackers. Furthermore TPF has similar performance to the best performing tracker(s) for the other sequences except for P6, P12, P17 and P22. TPF_3^* achieves an overall improvement of 23%, 15%, 8%, 23%, 21%, 27%, 13% and 17% in O_G in comparison to the individual trackers ST, AFT, LOT, IVT, FCT, AMS, L1T and LSST respectively. Moreover, all other TPF configurations (TPF_4^*,...,TPF_8^*) also achieve better results as compared to all individual 8 trackers.

Figure 4.10 compares tracker accuracy using O_G values for selected sequences. The target in P2 changes its pose, causing AMS and L1T to lose the target between frames 80-90. Both failing trackers at this point are discarded by TPF. Performance of FCT and LSST drops gradually after frame 130 due to background clutter. The performance of TPF_7^* drops at frame 140 where the output is corrupted by low performing trackers (FCT and LSST), which are incorrectly determined as *on-target*, while other TPF configurations make use of the best performing trackers. For P4, all trackers lose the target between frames 60-110. All TPF configurations identify and achieve accuracy close to the best performing tracker (AFT). However TPF fails when all trackers are *off-target*. P21 undergoes scale changes as it moves away from the camera. O_G for all TPF configurations drops after frame 80, since all trackers remain *on-target* and form a single cluster. After frame 130, O_G for ST and IVT drops significantly since they cannot handle scale changes. However these trackers are discarded by TPF, while the performance for TPF_7^* further improves as a better performing tracker (L1T) is added in the framework.

Furthermore, to determine the statistical significance of the results obtained from the proposed approaches we use the two sample Student's t-test assuming unequal variances. From each of the three (fused trackers, non-weighted fusion and weighted fusion), we select the best (LOT, TPF_3^* and TPF_3^{**}, respectively) and worst performing methods (AMS, TPF_7^* and TPF_7^{**}, respectively) and compare results obtained from the fusion methods to the selected trackers. We

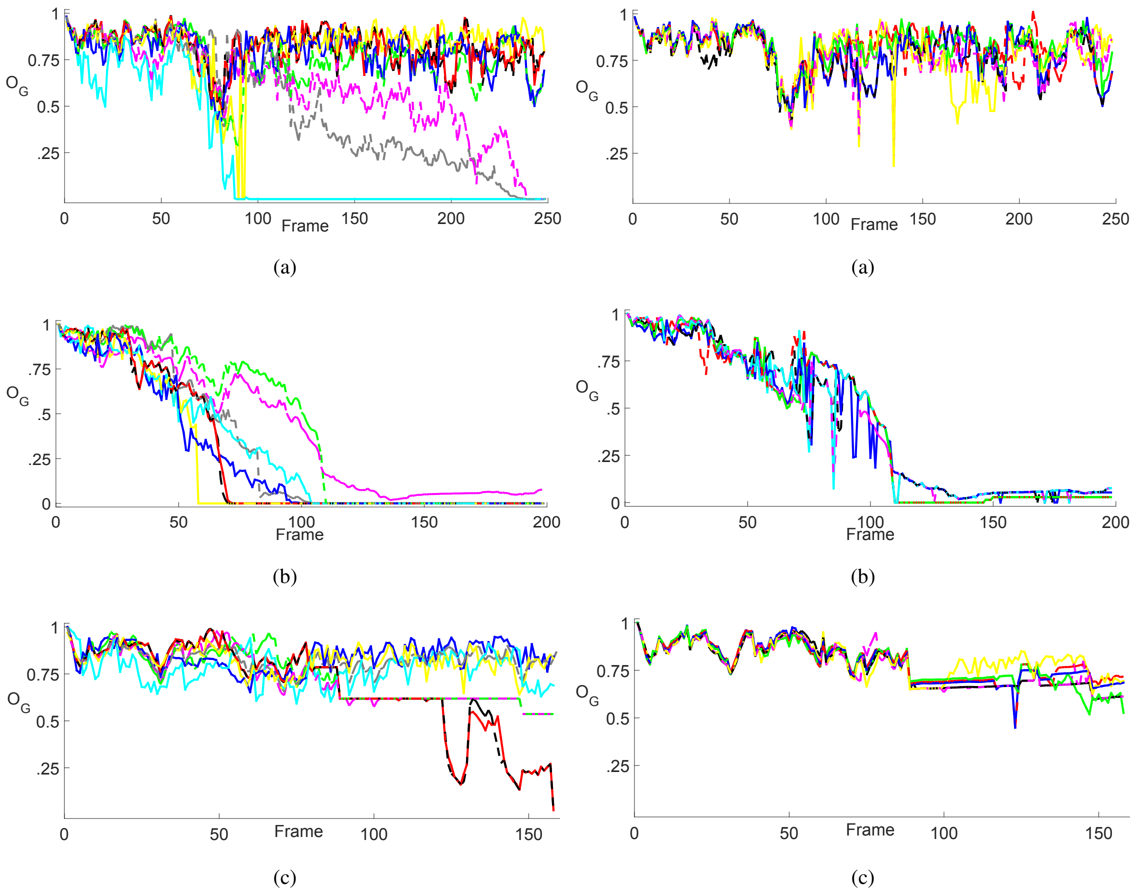


Figure 4.10: O_G scores for trackers and TPF configurations under analysis for selected sequences. (a) Students-P2, (b) CAVIAR-P4, (c) MITTraffic-P21. Left row: Trackers; —: ST; - - -: AFT; —: LOT; - - -: IVT; —: AMS; - - -: FCT; —: L1T; —: LSST. Right row: TPF configurations; —: TPF_3^*; - - -: TPF_4^*; —: TPF_5^*; - - -: TPF_6^*; —: TPF_7^*; - - -: TPF_8^*.

apply the significance test to the results obtained from TPF_3^*, TPF_7^*, TPF_3^{**} and TPF_7^{**} using the following experimental settings: the results from AMS and LOT are used as baseline, the total sample is the total number of sequences (i.e. twenty two) and 0.10 is used as the standard significance value.

The first test compares TPF_3^* with AMS and returns a p-value of 0.03 (less than the significance value) and rejects the null hypothesis H_0 in favour of TPF_3^* under the following hypothesis:

$$\begin{aligned} H_0 :& \quad \mu_{TPF_3^*} = \mu_{\text{AMS}}, \\ H_1 :& \quad \mu_{TPF_3^*} \neq \mu_{AMS}. \end{aligned} \tag{4.29}$$

where H_0 is rejected (H_1 is accepted) if TPF_3^* performs better than AMS. The same hypothesis tests TPF_3^* with LOT (by replacing μ_{AMS} with μ_{LOT} in Eq. 4.29) and a p-value of 0.48 accepts H_0.

The following hypothesis:

$$\begin{aligned} H_0 &: \quad \mu_{TPF_7^*} = \mu_{\text{AMS}}, \\ H_1 &: \quad \mu_{TPF_7^*} \neq \mu_{AMS}. \end{aligned} \tag{4.30}$$

compares TPF_7^* with AMS and a p-value of 0.12 accepts H_0 in favour of AMS. Replacing μ_{AMS} with μ_{LOT} in Eq. 4.30, TPF_7^* is then compared with LOT and a p-value of 0.89 accepts the H_0.

To compare the weighted fusion methods (TPF_3^{**} and TPF_7^{**}) against AMS and LOT, similar hypothesis are used where TPF_3^* and TPF_7^* are replaced by TPF_3^{**} and TPF_7^{**} in Eqs. 4.29 and 4.30, respectively. H_0 is rejected (accepted) with a p-value of 0.02 (0.47) when TPF_3^{**} is compared with AMS (LOT) and is accepted with p-values of 0.13 and 0.93 when TPF_7^{**} is compared with AMS and LOT, respectively.

The results from the significance test highlight the primary objective of the fusion framework i.e. to obtain better or equal tracking accuracy to the best tracker. The results for all four fusion methods indicate that they would achieve similar results to the best tracker (LOT) if additional experiments are performed. The results also highlight the second objective of the framework i.e. using a performance evaluation method to give lesser weight to bad performing trackers. H_0 is rejected when AMS is compared to TPF_3^* and TPF_3^{**} and is accepted with a low p-value (casting a doubt on the validity of H_0) in comparison to TPF_7^* and TPF_7^{**}.

4.8.2 Comparison of the fusion methods

Table 4.4 compares the two fusion methods. Results indicate that $TPF_3^{**}....TPF_8^{**}$ achieve similar accuracy to $TPF_3^*....TPF_8^*$ in terms of O_G, suggesting that including a performance-based weight to the fusion approach does not improve the overall tracking accuracy. The main reason for having similar results for both approaches highlights the fact that TPF selects only the *on-target* trackers instead of using results from all trackers. Since performance-weight is based on a state-background discrimination approach and C_t^* contains trackers that have a high spatial-correlation at each frame, the performance-weight of all trackers remain similar to each other. This indicates that the weight of a highly accurate tracker can be similar to a lesser accurate tracker since they share the same spatial region. Moreover, this also highlights the efficiency of the clustering approach that is able to correctly identify clusters of trackers that have a high spatio-temporal correlation over time.

4.8.3 Comparison with selected benchmark methods

Table 4.4 compares the TPF configurations and the related SOA. AvgF and SymT have been tested using the eight trackers. STR is the best for P1 and P10, achieving the best average results among the selected SOA approaches. KCF achieves the best results for P2, P3, P4, P5, P17 and P20. However, it is unable to handle occlusions as shown for P10, P12 and P22. SymT fails to determine a low performing tracker, hence reducing the overall tracking accuracy. It achieves good performance when most of the trackers are accurate as indicated by results for P7 and P15. TPF on the other hand is able to use the best performing trackers and the overall accuracy is not dependent on the percentage of good trackers. VTS performs relatively well and shows the best results for P8, P9 and P21. However, it fails for P1 and P10 due to occlusions, and for P17 and P22 due to similarly coloured background. Although the state-of-the-art approaches outperform some employed trackers (ST, IVT, FCT and AMS, see Table 4.4), TPF_3^* shows an overall improvement of 23%, 15%, 13%, 11% and 15% in O_G in comparison to AvgF, SymT, VTS, STR and KCF, respectively.

Sample tracking results for some sequences are shown in Figure 4.13 and Figure 4.14 where it can be seen that TPF correctly discards wrong trackers as they start to fail due to tracking challenges. For clarity we only present comparisons between TPF_3^* and the SOA. Examples in Figure 4.13(a)-(b) show that all trackers correctly follow the target at the beginning of the sequence. As target occlusions are more frequent, only STR is able to perform similarly to TPF_3^* as seen in frame 230 for P1 (Figure 4.13(a)) and frame 145 for P10 (Figure 4.13(b)). Figure 4.14(a) shows an example where none of trackers obtain accurate position estimations after an illumination change (frames 117 and 192) and the best trackers (KCF, STR and TPF_3^*) achieve low accuracy. Figure 4.14(b) depicts the situation where only TPF_3^* is able to adapt to changes in target scale and occlusions whereas all the compared trackers fail, as seen in frame 85.

4.9 Computational cost

Figure 4.11 presents the cost for the *trackers*, *tracker clustering* (Section 4.2) and *on-target cluster identification* (Section 4.3) in terms of average computational time. The cost of the fusion stage is negligible and therefore ignored. The framework has been implemented and tested on MATLAB2014b on an Intel(R) core(TM) i5-3570, 3.4GHz CPU with 8GB RAM on windows

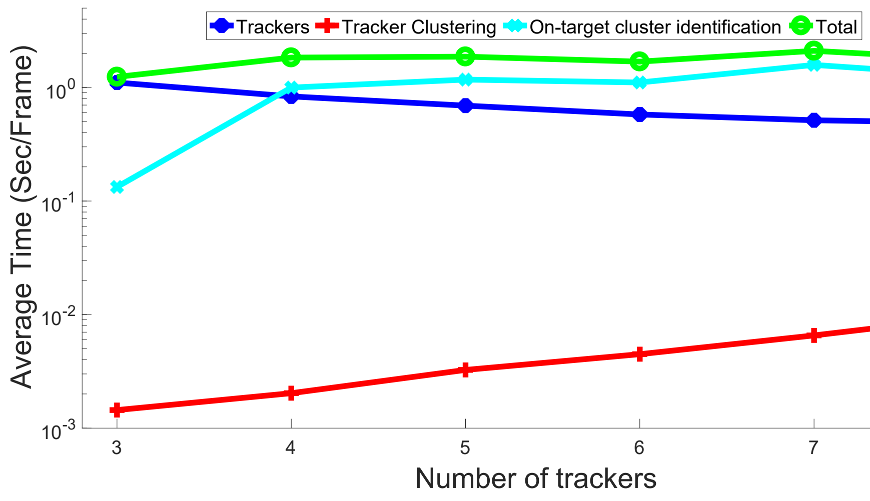


Figure 4.11: Average computational time for the stages of the proposed approach. For each configuration, the average is computed over the complete dataset and the total number of trackers.

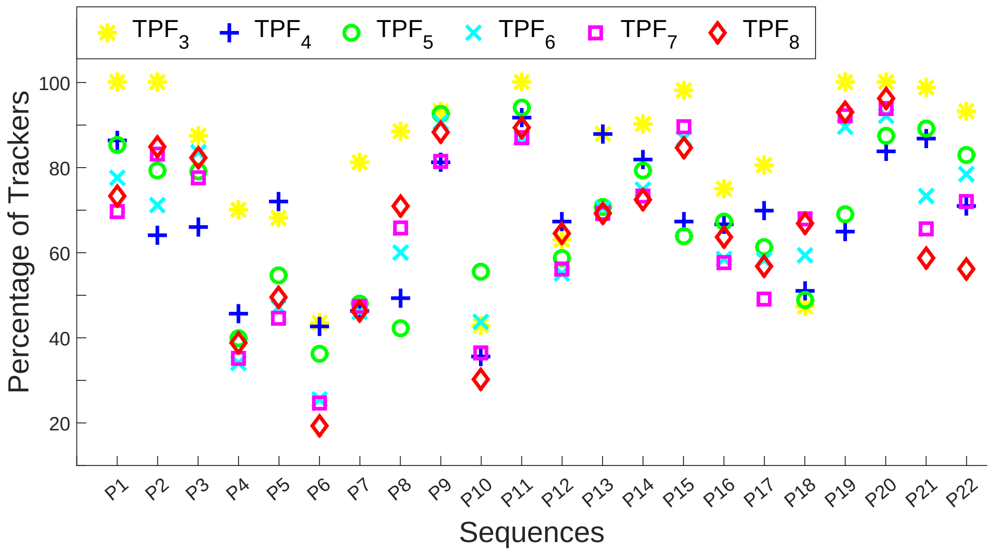


Figure 4.12: Percentage of trackers used by the proposed approach for different tracker combinations.

7 (64-bit). The cost of the *trackers* considers running in parallel the trackers to fuse and depends on the employed approaches, being heavily influenced by the slowest tracker (LOT). The computational time for *tracker clustering* slightly increases with the number of trackers. Since *on-target cluster identification* uses reverse-analysis, the computational time becomes dependent on the trackers in the *on-target* cluster $\mathbb{C}_t^*$ and the tracking challenges present in the sequence. This trend is also highlighted by the overall cost for the TPF configurations presented in Table 4.5, where TPF_3^* achieves the best computational cost. Figure 4.12 shows the average number of trackers used by TPF, highlighting its advantage to cluster trackers, and using only the ones

Table 4.5: Computational cost of the trackers and the proposed approach (TPF) measured as frames per second (FPS). Key - ST: Sparse Tracker, AFT: Adaptive Fragments based Tracker; LOT: Locally Orderless Tracker; IVT: Incremental Visual Tracker; FCT: Fast Compressive Tracker; AMS: Mean Shift Tracker; L1T: L1 Tracker; LSST: Least Soft-Threshold Squares Tracker.

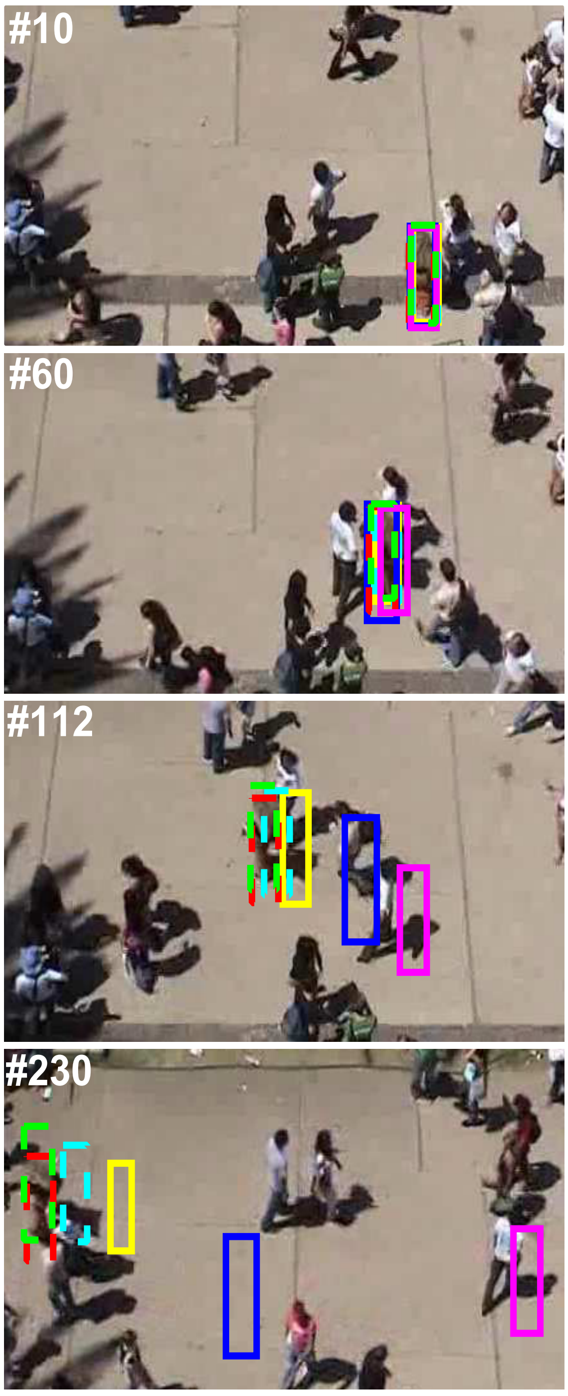
	Fused Trackers								Proposed approaches					
	ST [143]	AFT [42]	LOT [104]	IVT [110]	FCT [166]	AMS [102]	L1T [93]	LSST [142]	TPF_3^*	TPF_4^*	TPF_5^*	TPF_6^*	TPF_7^*	TPF_8^*
FPS	50.1±0.1	5.9±4.0	0.3±0.1	48.1±1.4	8.0±6.2	108.0±31.2	7.7±1.7	3.4±0.6	2.5±4.8	0.3±0.4	0.2±0.3	0.2±0.1	0.1±0.1	0.1±0.1

on-target for the various TPF combinations.

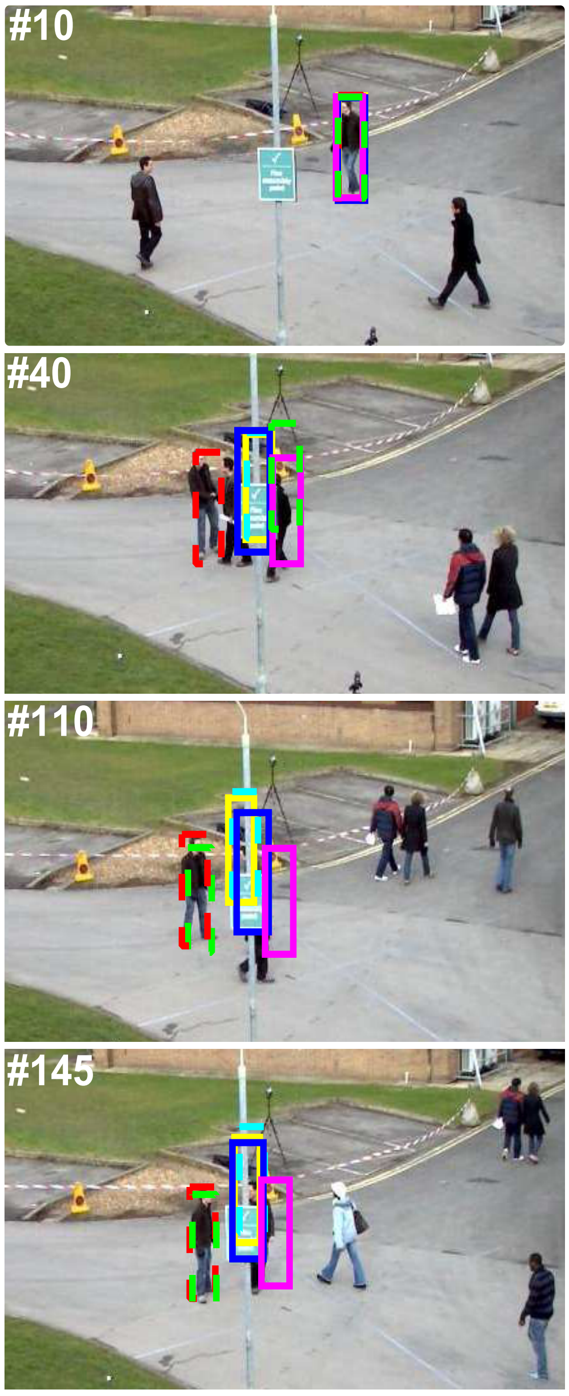
4.10 Summary

We presented an approach to dynamically select and combine the results of successful (i.e. *on-target*) trackers in a tracker-level fusion framework. The proposed approach determines relationships between trackers by analyzing the position and direction of their estimated states. These spatio-temporal features are combined to estimate pair-wise tracker correlation scores that determine clusters of similarly performing trackers over time. An adaptive online evaluator identifies the trackers that are *on-target* and propagates them over time until a split or merge of this group (cluster) of trackers is detected. The final target state is estimated by fusing the outputs from the trackers that are in the *on-target* cluster.

The framework is demonstrated using six different combinations that involve eight different trackers [42, 93, 102, 104, 110, 142, 143, 166] over a diverse dataset involving 22 sequences. The proposed approach of using two features (spatial overlap and temporal direction) within the clustering framework improves performance in comparison to only using either one of the features. Moreover, the proposed adaptive time-reversed evaluation improves the original approach [149] by using motion analysis and tracker performance to temporally update the reference frame. Finally, by using only the trackers within the *on-target* cluster for fusion, the proposed framework is able to outperform state-of-the-art methods and the combined trackers.



a



b

Figure 4.13: Sample tracking results for (a) Students-P1, (b) PETS-P10. - - -: TPF; - - -: STRUCK; —: VTS; - - -: SymT; —: AvgF; —: KCF.

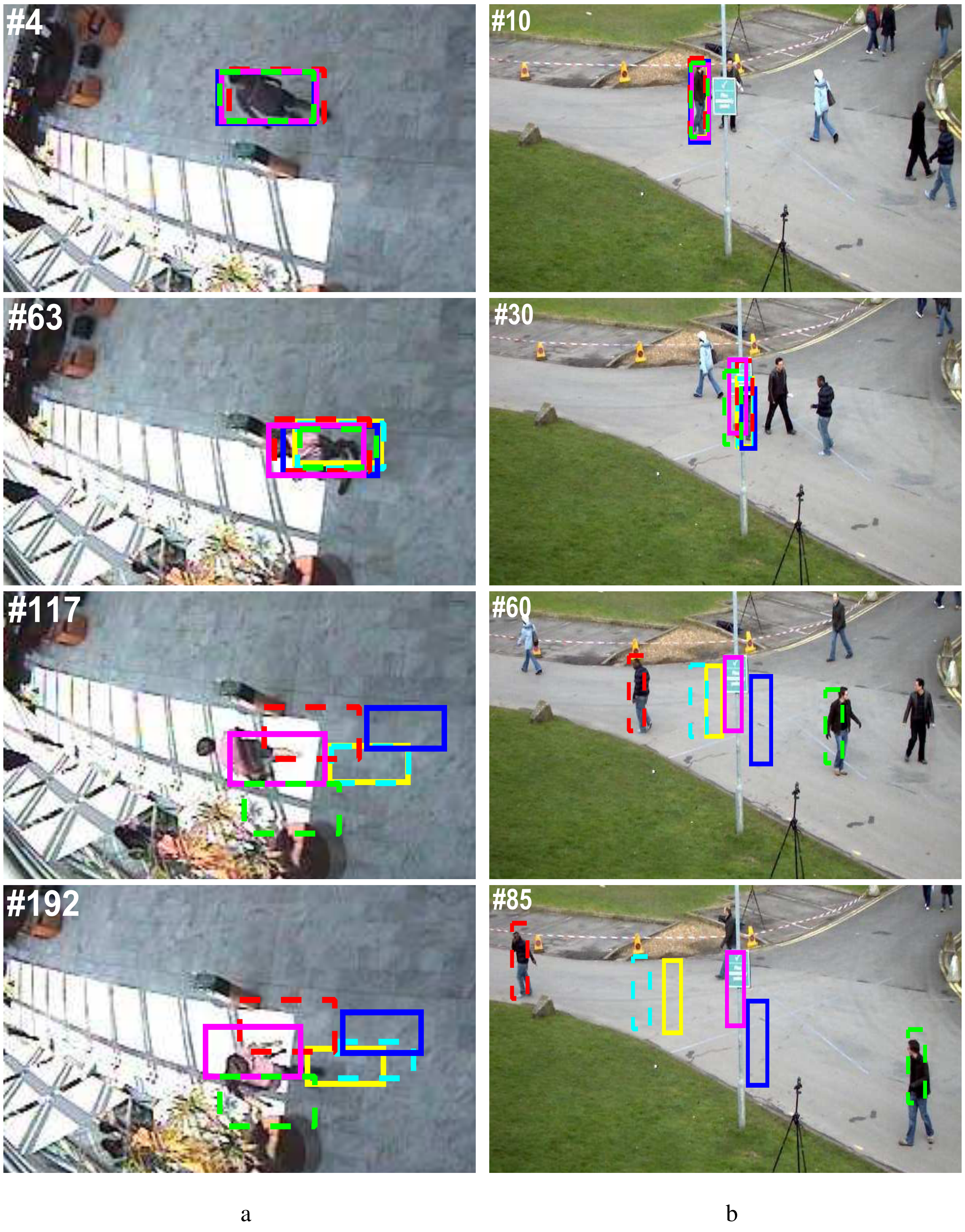


Figure 4.14: Sample tracking results for (a) CAVIAR-P4 and (b) PETS-P12. - - -: TPF; - - -: STRUCK; —: VTS; - - -: SymT; —: AvgF; —: KCF.

Chapter 5

Conclusions

5.1 Summary of achievements

In this thesis we present an approach to improve the overall tracking accuracy by fusing the outputs of multiple video trackers that track the same target. However, when combining multiple trackers within the same framework, some of the trackers may fail due to the challenges present within the video data. Failing trackers can cause the overall tracking accuracy to drop and furthermore, can be a computational burden within the fusion framework. In this thesis we therefore address different problems regarding tracker-level fusion within a single target tracking framework.

The *first* problem deals with the online performance evaluation of video trackers without being dependent upon the type of tracker. Combining trackers within a fusion framework may require the performance evaluation of each tracker. Performance evaluation helps in determining low-performing trackers that can either be given less weight or be discarded from the fusion. Having a performance evaluator dependent on the type of tracker may limit the fusion framework to specific trackers. We present an approach that is tracker-independent and determines feature descriptors using tracking outputs only to evaluate the tracking performance. The method uses a state-background discrimination approach to determine the similarity (dissimilarity) of the two regions, which indicates low (high) performance of the tracker.

The *second* problem involves using tracking performance scores that have varying range of values for different datasets or trackers. We model the tracking performance score as a time series and employ a forecasting function that estimates future values of the performance score. The difference between the original and forecast values returns the forecast error signal which has a uniform range of values for the different datasets and trackers used to test the approach. Then, using the forecast error signal we determine time instants when the tracker fails.

Our *third* problem is concerned with finding the trackers (from a group of trackers) that are on the correct target within the fusion framework. The current SOA approaches to tracker-level fusion either use all trackers for fusion or select only the best performing tracker to estimate the final target state. Our clustering approach to tracker fusion enables us to determine different groups of trackers that may form when some trackers fail due to challenges within the video data. We achieve this by estimating the spatio-temporal relationships between the outputs of the trackers. Then, we use an adaptive time-reversed evaluation approach to identify the *on-target* cluster i.e. the cluster which follows the correct target. We then fuse the outputs from the selected trackers (part of the *on-target* cluster) to estimate the final target state. To further reduce the computational load of a tracker-level fusion framework, we propagate the *on-target* cluster over time until trackers leave or join this cluster. This allows us to reduce the computational cost during the performance evaluation and the fusion by using trackers only within the *on-target* cluster.

The computational cost of the original time-reversed evaluation approach [149] is our *fourth* problem. The method evaluates tracking performance by comparing results of the tracker with that of another tracker running in time-reversed direction at a specified reference frame (i.e. where the tracker is known to be on target). Using the first frame of the sequence as the reference frame imposes a heavy computational cost, making the method [149] infeasible for longer sequences. We address this problem by adaptively moving the reference frame forward in time by using information based on the motion of the target state and the performance of the trackers.

5.2 Future work

The following section presents the possible future research directions of the thesis.

1. The method of detecting tracking failure employs a state-background discrimination approach. The position for the background is estimated in the one time step ahead in the future. This can be changed to estimating the position of the background in the future within a certain temporal window. This can allow the method to predict tracking failure in

advance and help the tracker to self-correct when the failure occurs.

2. The method of detecting tracking failure employs features that are dependent upon the colour space, hence making the approach limited to visual challenges such as background clutter, sudden illumination changes or fast target motion. Combining other features such as gradient-based or those based on motion patterns can improve the robustness of the method towards such challenges.

3. The method of detecting tracking failure employs an experimentally determined threshold. The experimental results highlights that the threshold can be applied to different trackers and different sequences, however, a threshold-free approach is desirable. Determining the model of the forecast error signal or using methods such as the forecast tracking signal [50] can be employed to remove dependence over thresholds. A forecast tracking signal monitors the forecasts made with respect to the originals and indicates when there are unexpected changes within the forecast values.

4. The method of detecting tracking failure also presents an approach to measure tracking accuracy and robustness (failure) without using any ground truth data. This information can be combined to generate a new measure similar to Combined Tracking Performance Score (CoTPS) [98]. CoTPS is an offline performance evaluation method (i.e. based on GT) and combines tracking accuracy and robustness to generate a unified score for measuring tracking performance.

5. The clustering-based tracker fusion approach has been tested on a single target tracking framework. The method can also be extended to a multi-target tracking (MTT) framework where information from multiple trackers can be employed to improve tracking accuracy. However, such a method will also need to handle the various issues of a MTT framework that include data association as well as detection, initiation and termination of targets in a detection-based MTT method.

6. The proposed fusion approach can also be applied to a convolutional network-based (deep learning) framework. A single strong tracker can be built by fusing the outputs of multiple trackers that use features extracted from different layers of the network [107]. However, combining outputs from all trackers may cause the overall tracking accuracy to drop due to failing trackers. The proposed fusion approach first employs a clustering-based method to determine the different groups of trackers and then identifies the *on-target* group using an

online performance evaluator. The clustering-based method combined with a performance evaluator can be used to further improve the tracking accuracy of the deep learning-based fusion method.

7. The method for determining tracking performance based on reverse-tracking employs an adaptive reference frame. Since the method is based on motion analysis, the approach of moving the reference frame forward in time cannot differentiate between two situations i.e. (i) the tracker is *on-target* and the target is static; (ii) the tracker drifts from the target and gets locked onto a static background region. The performance of the approach of moving the reference frame forward in time can be further improved by employing methods of foreground-background discrimination to differentiate between the two situations.

8. Based on statistical analysis performed in Section 4.6.2, further experiments can be performed to determine the optimal value of ω (Eq. 4.7) that may help to improve the performance of the clustering method.

Appendix A

Evaluation datasets

A.1 Introduction

To evaluate the approaches discussed in the thesis we use three different datasets selected from the SOA. The datasets cover both indoor and outdoor sequences containing multiple tracking challenges that include partial and full occlusions, background clutter, target rotation including changes in scale, pose, motion and illumination of the target and include five target types, namely person, faces, vehicles, animals and others.

For the first dataset D1, 22 sequences (3580 frames) was considered from the following datasets: Students[1], CAVIAR[2], PETS (2009[3] and 2001[4]), LTDT[5], TRECVID2009[6], MIT Traffic[7], David[8] and AVSS2007[9]. The second dataset OTB that includes 20 sequences (6600 frames) has been taken from the Object Tracking Benchmark [155]. Sequences *David* (300:500), *Doll* (1:500), *Girl* (1:210), *Liquor* (1:750) and *Woman* (1:150) (first frame:last frame), are used with reduced number of frames since the tracker fails to recover the target after this point. Dataset VOT, 25 sequences (10200 frames) has been used from the VOT2014 challenge [10].

We use the dataset D1 for testing TPF and determining the optimal parameters for DTFF.

[1] http://graphics.cs.ucy.ac.cy/research/downloads/crowd-data
[2] http://groups.inf.ed.ac.uk/vision/CAVIAR/CAVIARDATA1/
[3] http://www.cvg.reading.ac.uk/PETS2009
[4] http://www.cvg.reading.ac.uk/slides/pets.html
[5] http://www.micc.unifi.it/LTDT2014
[6] http://trecvid.nist.gov/trecvid.data.html#tv09
[7] http://www.ee.cuhk.edu.hk/~xgwang/HBM.html
[8] http://www.cs.toronto.edu/~dross/ivt/
[9] http://www.avss2007.org/
[10] http://www.votchallenge.net/vot2014/dataset.html

DTFF is then tested over dataset OTB, while further validation is performed using the VOT2014 challenge dataset. Table A.1, A.2 and A.3 describes the sequences for datasets D1, OTB and VOT, respectively. The target initialization for D1, OTB and VOT are included in Figure A.1, Figure A.2 and Figure A.3, respectively.

A.2 Tabular summary

Table A.1: Sequences used in the experiments. KEY - BC: background clutter; P: pose variations; O: occlusions; I: illumination variations; S: scale variations; M: motion variations.

Dataset		Sequence name	Target Class	Size		Total Frames	Challenges
				Target	Frame		
Students	P1	University Students	Person	21 x 75	720 x 576	250	BC, P, O, I
	P2			22 x 69		250	BC, P, I
	P3			25 x 61		165	P, I
CAVIAR	P4	Browse_WhileWaiting1		50 x 24	384 x 288	200	P, I
	P5	OneLeaveShopReenter1Front		16 x 56		195	P, I, S, BC
	P6	OneLeaveShopReenter2front		14 x 50		300	P, I, S, BC
	P7	ThreePastShop2cor		56 x 142		170	P, I, S
PETS	P8	S2.L2 walking		14 x 50	768 x 576	140	P, I, S, BC
	P9	PETS2001 Dataset 1	Vehicle	56 x 142		150	P, I, S
	P10	S2.L1 walking	Person	22 x 68		150	BC, I,S
	P11	PETS2001 Dataset 1	Vehicle	72 x 56		180	S, M
	P12	S2.L1 walking	Person	72 x 56		90	O, M
	P13			72 x 56		150	M, O, P
	P14			72 x 56		110	S, M
LTDT	P15	NissanSkylineChase	Vehicle	37 x 21	640 x 275	300	I, M, S
David	P16	David Indoor	Face	91 x 116	320 x 240	130	I, S, M, BC
AVSS2007	P17	Abandoned baggage	Person	60 x 240	720 x 576	200	P, I
TRECVID	P18	MCTTR0205a		72 x 226	720 x 576	50	P, I, BC
	P19			64 x 204		40	P, I, BC
	P20			64 x 204		40	I, M, BC
MIT Traffic	P21	MV2_001	Vehicle	34 x 26	720 x 480	160	I, S, M
	P22	MV2_006		70 x 36	720 x 576	160	O, I, M

Table A.2: Selected sequences from the Object Tracking Benchmark (OTB) dataset [155] used in the experiments. KEY - I: illumination variation; BC: background clutter; S: scale variations; O: occlusions; P: pose changes; MB: motion blur.

Sequence name	Target Class	Size		Total Frames	Challenges
		Target	Frame		
CarDark	Vehicle	29 x 23	320 x 240	300	I, BC
CarScale		42 x 26	640 x 272	250	S, O, P
Couple	Person	25 x 62	320 x 240	140	S, P, BC
Crossing		17 x 50	360 x 240	120	S, P, BC
David	Face	64 x 78	320 x 240	200	I, S, O
David3	Person	35 x 131	640 x 480	230	O, P, BC
Doll	Other	32 x 73	400 x 300	500	I, S, O, P
FaceOcc1	Face	114 x 162	352 x 288	890	O
Girl	Person	31 x 45	128 x 96	210	S, O, P
Jogging		25 x 101	352 x 288	305	O, P
Liquor	Other	73 x 210	640 x 480	750	I, S, O, MB, P, BC
MotorRolling	Vehicle	122 x 125	640 x 360	165	I, S, MB, P, BC
MountainBike		67 x 56	640 x 360	230	P, BC
Singer1	Person	87 x 290	624 x 352	350	I, S, O, P
Singer2		67 x 122	624 x 352	365	I, P, BC
Subway		19 x 51	352 x 288	175	O, BC
Tiger1	Other	76 x 84	640 x 480	355	I, O, MB, P
Walking	Person	24 x 79	768 x 576	410	S, O
Walking2		31 x 115	384 x 288	500	S, O
Woman		21 x 95	352 x 288	150	I, S, O, MB, P

Table A.3: Sequences from the Visual Object Tracking (VOT) 2014 challenge [73] used in the experiments. KEY - BC: background clutter; P: pose variations; O: occlusions; I: illumination variations; S: scale variations; M: motion variations; CM: camera motion.

Sequence name	Target Class	Size		Total Frames	Challenges
		Target	Frame		
ball	Other	45 x 45	320 x 240	600	S, M, CM
basketball	Person	31 x 113	576 x 432	725	O, M, CM, S, BC
bicycle		18 x 48	320 x 240	270	O, M, S, CM
bolt		3 x 68	640 x 360	350	M, CM
car	Vehicle	43 x 27	640 x 272	250	O, M, S, CM
david	Face	82 x 96	320 x 240	770	I, CM, S, M
diving	Person	37 x 162	400 x 224	220	M, S, CM
drunk	Vehicle	105 x 91	508 x 336	1210	M, S
fernando	Animal	83 x 217	640 x 480	290	O, M, I, CM, S, BC
fish1		46 x 8	460 x 259	435	M, S, CM
fish2		78 x 81	640 x 360	310	O, M, S, CM, BC
gymnastics	Person	41 x 124	320 x 180	205	M, S, CM
hand1	Other	43 x 45	320 x 240	245	S, M
hand2		47 x 53	320 x 240	265	S, M
jogging	Person	26 x 102	352 x 288	305	O, S, M, CM
motocross	Vehicle	106 x 145	640 x 360	165	S, M, CM, BC
polarbear	Animal	50 x 71	640 x 360	370	S, M, CM
skating	Person	35 x 88	768 x 360	400	S, M, O, CM, I
sphere	Other	87 x 90	480 x 360	200	S, M, I, CM
sunshade	Face	37 x 51	352 x 288	170	S, M, I, CM
surfing	Person	19 x 47	320 x 240	280	M, I, CM
torus	Other	49 x 50	320 x 240	265	M, S
trellis	Face	68 x 73	320 x 240	570	S, M, I, CM
tunnel	Person	54 x 87	360 x 480	730	S, M, I, CM
woman		29 x 103	352 x 288	600	S, M, O, CM

A.3 Target initializations

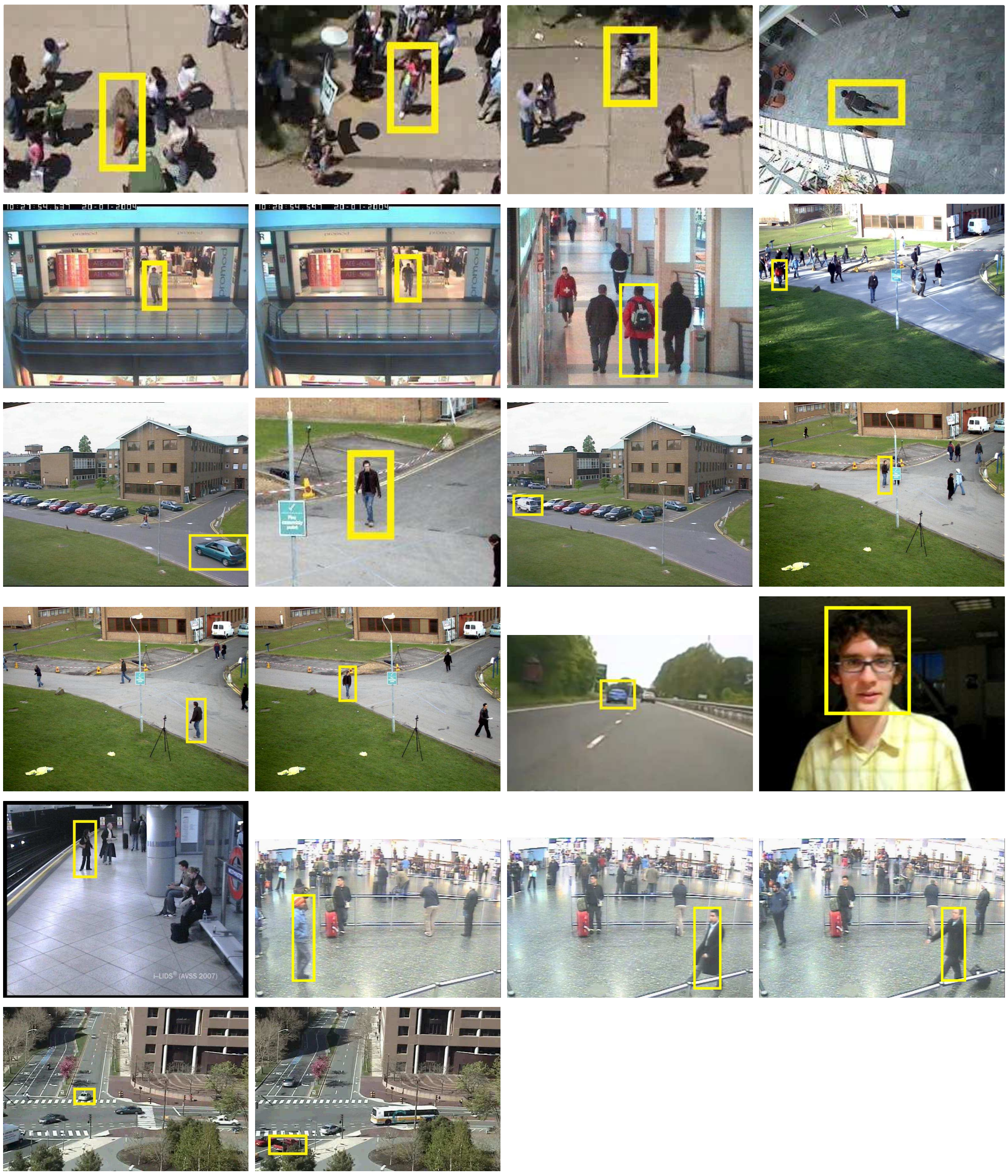


Figure A.1: Target initializations for dataset D1.

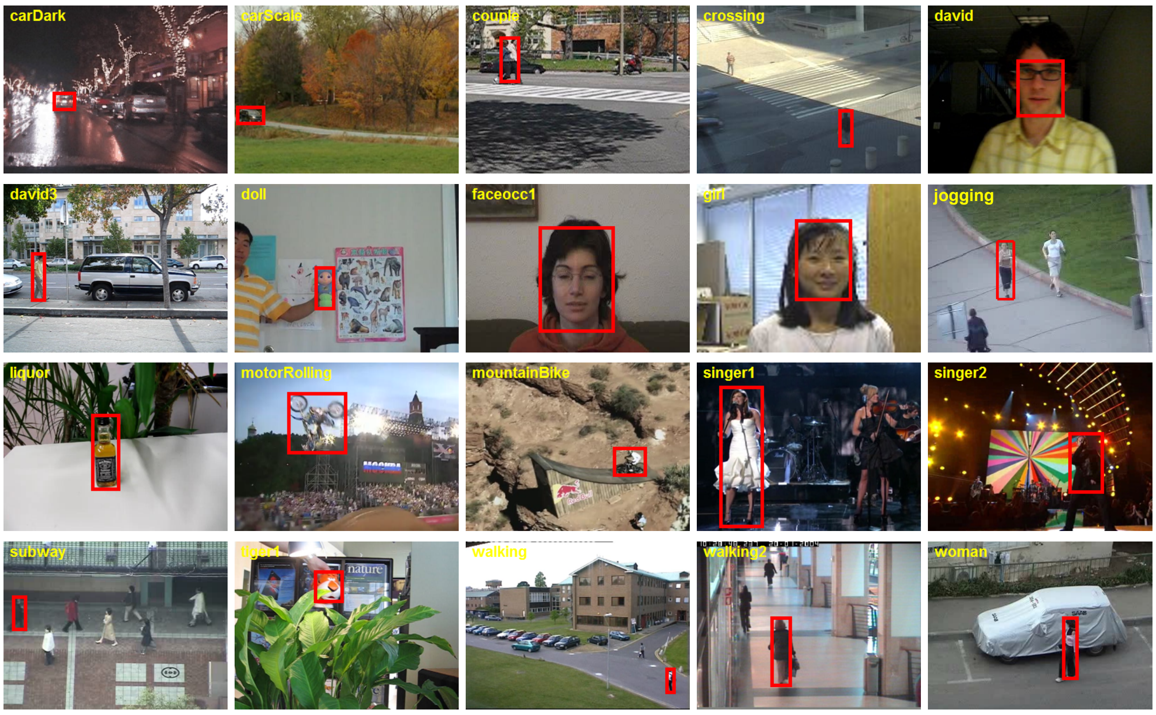


Figure A.2: Target initializations for dataset OTB

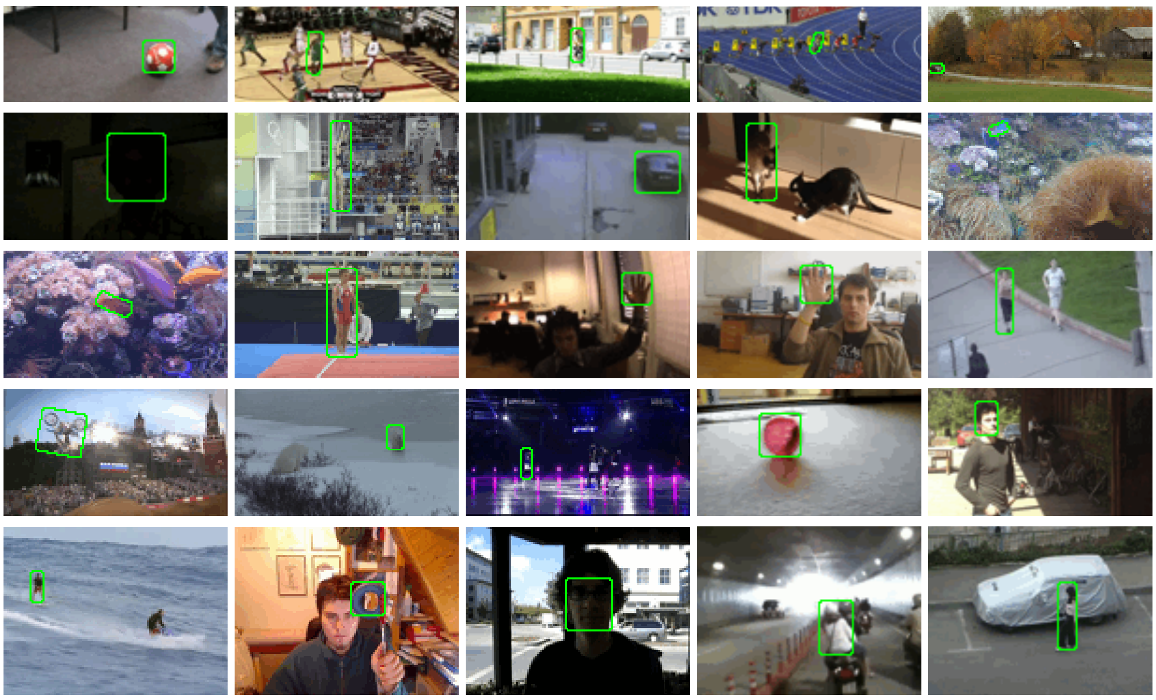


Figure A.3: Target initializations for dataset VOT2014

Bibliography

[1] S. Abosedra and H. Baghestani. On the predictive accuracy of crude oil futures prices. *Energy Policy*, 32(12):1389 – 1393, Aug 2004.

[2] A. Adam, E. Rivlin, and I. Shimshoni. Robust fragments-based tracking using the integral histogram. In *IEEE International Conference on Computer Vision and Pattern Recognition*, pages 798–805, 17-22 Jun 2006.

[3] N. Anjum and A Cavallaro. Multifeature object trajectory clustering for video analysis. *IEEE Transactions on Circuits and Systems for Video Technology*, 18(11):1555–1564, Nov 2008.

[4] M. S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2):174–188, Feb 2002.

[5] S. Avidan. Ensemble tracking. In *IEEE International Conference on Computer Vision and Pattern Recognition*, pages 494–501, 20-25 Jun 2005.

[6] B. Babenko, M.H. Yang, and S. Belongie. Visual tracking with online multiple instance learning. In *IEEE International Conference on Computer Vision and Pattern Recognition*, pages 983–990, 20-25 Jun 2009.

[7] V. Badrinarayanan, P. Perez, F. Le Clerc, and L. Oisel. Probabilistic color and adaptive multi-feature tracking with dynamically switched priority between cues. In *IEEE International Conference on Computer Vision*, pages 1–8, 14-21 Oct 2007.

[8] Q. Bai, Z. Wu, S. Sclaroff, M. Betke, and C. Monnier. Randomized ensemble tracking. In *IEEE International Conference on Computer Vision*, pages 2040–2047, 1-8 Dec 2013.

[9] C. Bailer, A. Pagani, and D. Stricker. A superior tracking approach: Building a strong tracker through fusion. In *Proceedings of European Conference on Computer Vision*, pages 170–185, 6-12 Sep 2014.

[10] L. Bertinetto, J. Valmadre, S. Golodetz, O. Miksik, and P. H. S. Torr. Staple: Complementary learners for real-time tracking. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1401–1409, 27-30 Jun 2016.

[11] S. Bianco, G. Ciocca, P. Napoletano, and R. Schettini. An interactive tool for manual, semi-automatic and automatic video annotation. *Computer Vision and Image Understanding*, 131:88–99, Feb 2015.

[12] E. Bingham and H. Mannila. Random projection in dimensionality reduction: Applications to image and text data. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, pages 245–250, 26-29 Aug 2001.

[13] S. Birchfield. Elliptical head tracking using intensity gradients and color histograms. In *IEEE International Conference on Computer Vision and Pattern Recognition*, pages 232–237, 25 Jun 1998.

[14] T.A. Biresaw, M.S. Alvarez, and C.S. Regazzoni. Online failure detection and correction for bayesian sparse feature-based object tracking. In *IEEE International Conference on Advanced Video and Signal based Surveillance*, pages 320–324, 30 Aug-2 Sep 2011.

[15] T.A. Biresaw, A. Cavallaro, and C.S. Regazzoni. Correlation-based self-correcting tracking. *Neurocomputing*, 152(1):345–358, Mar 2015.

[16] T.A. Biresaw, A. Cavallaro, and C.S. Regazzoni. Tracker-level fusion for robust bayesian visual tracking. *IEEE Transactions on Circuits and Systems for Video Technology*, 25(5):776–789, May 2015.

[17] T.A. Biresaw, T. Nawaz, J. Ferryman, and A. Dell. ViTBAT: Video tracking and behavior annotation tool. In *IEEE International Conference on Advanced Video and Signal based Surveillance*, 24-26 Aug 2016.

[18] J. Black, T. Ellis, and P. Rosin. A novel method for video tracking performance evaluation. In *Joint IEEE International Workshop on Visual Surveillance and Performance Evaluation of Tracking and Surveillance*, pages 125–132, Oct 2003.

[19] M. B. Blaschko and C. H. Lampert. Learning to localize objects with structured output regression. In *Proceedings of European Conference on Computer Vision*, pages 2–15, 12-18 Oct 2008.

[20] F. Bousetouane, L. Dib, and H. Snoussi. Improved mean shift integrating texture and color features for robust real time object tracking. *The Visual Computer*, 29(3):155–170, Mar 2013.

[21] T. Bouwmans. Traditional and recent approaches in background modeling for foreground detection: An overview. *Computer Science Review*, 11:31–66, May 2014.

[22] G.E.P. Box, G.M. Jenkins, and G.C. Reinsel. *Time series analysis: Forecasting and control*. Wiley, 4 edition, Jun 2008.

[23] P. Brasnett, L. Mihaylova, D. Bull, and N. Canagarajah. Sequential monte carlo tracking by fusing multiple cues in video sequences. *Image and Vision Computing*, 25(8):1217 – 1227, Aug 2007.

[24] C. Chang, R. Ansari, and A. Khokhar. Multiple object tracking with kernel particle filter. In *IEEE International Conference on Computer Vision and Pattern Recognition*, pages 566–573, 20-25 Jun 2005.

[25] D.P. Chau, J. Badie, F. Brémond, and M. Thonnat. Online tracking parameter adaptation based on evaluation. In *IEEE International Conference on Advanced Video and Signal based Surveillance*, pages 189–194, 27-30 Aug 2013.

[26] D.P. Chau, F. Bremond, and M. Thonnat. Online evaluation of tracking algorithm performance. In *International Conference on Crime Detection and Prevention*, pages 1–6, 3 Dec 2009.

[27] D.P. Chau, F. Bremond, M. Thonnat, and E. Corve. Robust mobile object tracking based on multiple feature similarity and trajectory filtering. In *The International Conference on Computer Vision Theory and Applications*, pages 569–574, 5-7 Mar 2011.

[28] D.P. Chau, M. Thonnat, F. Bremond, and E. Corve. Online parameter tuning for object tracking algorithms. *Image and Vision Computing*, 32(4):287–302, Apr 2014.

[29] K. Chaudhuri, Y. Freund, and D. Hsu. A parameter-free Hedging algorithm. In *International Conference on Neural Information Processing Systems*, pages 297–305, 5-8 Dec 2009.

[30] M. Cimpoi, S. Maji, and A. Vedaldi. Deep filter banks for texture recognition and segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3828–3836, 7-12 Jun 2015.

[31] R.T. Collins, Y. Liu, and M. Leordeanu. Online selection of discriminative tracking features. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(10):1631–1643, Oct 2005.

[32] D. Comaniciu, V. Ramesh, and P. Meer. Kernel-based object tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(5):564–577, May 2003.

[33] J.H. Conway and R Guy. *The Book of Numbers*. Copernicus, Springer Science and Business Media, 1 edition, 1996.

[34] M. Danelljan, G. Hager, F. S. Khan, and M. Felsberg. Accurate scale estimation for robust visual tracking. In *Proceedings of the British Machine Vision Conference*, 1-5 Sep 2014.

[35] M. Danelljan, G. Hager, F. S. Khan, and M. Felsberg. Learning spatially regularized correlation filters for visual tracking. In *IEEE International Conference on Computer Vision*, pages 4310–4318, 13-16 Dec 2015.

[36] M. Danelljan, G. Hager, F. S. Khan, and M Felsberg. Discriminative scale space tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2016.

[37] M. Danelljan, A. Robinson, F. S. Khan, and M. Felsberg. Beyond correlation filters: Learning continuous convolution operators for visual tracking. In *Proceedings of European Conference on Computer Vision*, pages 472–488, 11-14 Oct 2016.

[38] A. Doulamis. Dynamic tracking re-adjustment: a method for automatic tracking recovery in complex visual environments. *Multimedia Tools and Applications*, 50(1):49–73, Oct 2010.

[39] S. Duffner, J. Odobez, and E. Ricci. Dynamic partitioned sampling for tracking with discriminative features. In *Proceedings of the British Machine Vision Conference*, 7-10 Sep 2009.

[40] A. Elgammal, D. Harwood, and L. Davis. Non-parametric model for background subtraction. In *Proceedings of European Conference on Computer Vision*, pages 751–767, 26 Jun-1 Jul 2000.

[41] C.E. Erdem, B. Sankur, and A.M. Tekalp. Performance measures for video object segmentation and tracking. *IEEE Transactions on Image Processing*, 13(7):937–951, Jul 2004.

[42] E. Erdem, S. Dubuisson, and I. Bloch. Fragments based tracking with adaptive cue integration. *Computer Vision and Image Understanding*, 116(7):827 – 841, Jul 2012.

[43] E. Erdem, S. Dubuisson, and I. Bloch. Visual tracking by fusing multiple cues with context-sensitive reliabilities. *Pattern Recognition*, 45(5):1948–1959, May 2012.

[44] H. Fan and H. Ling. SANet: Structure-aware network for visual tracking. *CoRR*, abs/1611.06878, 2016.

[45] J. Fan, X. Shen, and Y. Wu. What are we tracking: A unified approach of tracking and recognition. *IEEE Transactions on Image Processing*, 22(2):549–560, Feb 2013.

[46] M. Felsberg. Enhanced distribution field tracking using channel representations. In *IEEE International Conference on Computer Vision Workshops*, pages 121–128, 1-8 Dec 2013.

[47] P. Gabriel, J. B. Hayet, J. Piater, and J. Verly. Object tracking using color interest points. In *IEEE International Conference on Advanced Video and Signal Based Surveillance*, pages 159–164, 15-16 Sep 2005.

[48] J. Gao, T. Zhang, X. Yang, and C. Xu. Deep relative tracking. *IEEE Transactions on Image Processing*, PP(99):1–1, 2017.

[49] Y. Gao, R. Ji, L. Zhang, and A. Hauptmann. Symbiotic tracker ensemble towards a unified tracking framework. *IEEE Transactions on Circuits and Systems for Video Technology*, 24(7):1122–1131, Jul 2014.

[50] J.G. Gooijer and R.J. Hyndman. 25 years of time series forecasting. *International Journal of Forecasting*, 22(3):443 – 473, Jul 2006.

[51] H. Grabner, C. Leistner, and H. Bischof. Semi-supervised on-line boosting for robust tracking. In *Proceedings of European Conference on Computer Vision*, pages 234–247, 12-18 Oct 2008.

[52] G.D. Hager and P.N. Belhumeur. Efficient region tracking with parametric models of geometry and illumination. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(10):1025–1039, Oct 1998.

[53] P.R. Halmos. *Naive Set Theory*. Springer Science and Business Media, 1960.

[54] B. Han, Y. Zhu, D. Comaniciu, and L.S. Davis. Visual tracking by continuous density propagation in sequential bayesian filtering framework. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(5):919–930, May 2009.

[55] Z. Han, Q. Ye, and J. Jiao. Online feature evaluation for object tracking using kalman filter. In *International Conference on Pattern Recognition*, pages 1–4, 8-11 Dec 2008.

[56] Z. Han, Q. Ye, and J. Jiao. Combined feature evaluation for adaptive visual object tracking. *Computer Vision and Image Understanding*, 115(1):69 – 80, Jan 2011.

[57] S. Hare, S. Golodetz, A. Saffari, V. Vineet, M. M. Cheng, S.L. Hicks, and P.H.S. Torr.

Struck: Structured output tracking with kernels. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(10):2096–2109, Oct 2016.

[58] S. Hare, A. Saffari, and P.H.S. Torr. Struck: Structured output tracking with kernels. In *IEEE International Conference on Computer Vision*, pages 263–270, 6-13 Nov 2011.

[59] M. Heber, M. Godec, M. Ruther, P. Roth, and H. Bischof. Segmentation-based tracking by support fusion. *Computer Vision and Image Understanding*, 117(6):573–586, Jun 2013.

[60] D. Held, S. Thrun, and S. Savarese. Learning to track at 100 fps with deep regression networks. In *Proceedings of European Conference on Computer Vision*, pages 749–765, 11-14 Oct 2016.

[61] C.K. Heng, S. Yokomitsu, Y. Matsumoto, and H. Tamura. Shrink boost for selecting multi-lbp histogram features in object detection. In *IEEE International Conference on Computer Vision and Pattern Recognition*, pages 3250–3257, 23-28 Jun 2012.

[62] J.F. Henriques, R. Caseiro, P. Martins, and J. Batista. High-speed tracking with kernelized correlation filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(3):583–596, Mar 2015.

[63] C. C. Holt. Forecasting seasonals and trends by exponentially weighted moving averages. *International Journal of Forecasting*, 20(1):5–10, Jan 2004.

[64] W. Hu, X. Zhou, W. Li, W. Luo, X. Zhang, and S. Maybank. Active contour-based visual tracking by integrating colors, shapes, and motions. *IEEE Transactions on Image Processing*, 22(5):1778–1792, May 2013.

[65] R.J. Hyndman and A.B. Koehler. Another look at measures of forecast accuracy. *International Journal of Forecasting*, 22(4):679–688, Oct 2006.

[66] D-S. Jang, S-W. Jang, and H. Choi. 2D human body tracking with structural kalman filter. *Pattern Recognition*, 35(10):2041–2049, Oct 2002.

[67] F. Jiang, Y. Wu, and A.K. Katsaggelos. A dynamic hierarchical clustering method for trajectory-based unusual video event detection. *IEEE Transactions on Image Processing*, 18(4):907–913, Apr 2009.

[68] H. Jiang, J. Li, D. Wang, and H. Lu. Multi-feature tracking via adaptive weights. *Neurocomputing*, 207:189–201, Sep 2016.

[69] Z. Kalal, K. Mikolajczyk, and J. Matas. Forward-backward error: Automatic detection of tracking failures. In *International Conference on Pattern Recognition*, pages 2756–2759, 23-26 Aug 2010.

[70] R.G. Kavasseri and K. Seetharaman. Day-ahead wind speed forecasting using f-arima models. *Renewable Energy*, 34(5):1388–1393, May 2009.

[71] B. Khaleghi, A. Khamis, O.F. Karray, and N.S. Razavi. Multisensor data fusion: A review of the state-of-the-art. *Information Fusion*, 14(1):28–44, Jan 2013.

[72] M. Kristan, J. Matas, A. Leonardis, T. Vojir, R. Pflugfelder, G. Fernandez, G. Nebehay, F. Porikli, and L. Cehovin. A novel performance evaluation methodology for single-target trackers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(11):2137–2155, Nov 2016.

[73] M. Kristan, R. Pflugfelder, A. Leonardis, et al. The visual object tracking VOT2014 challenge results. In *Proceedings of European Conference on Computer Vision, Workshop on Visual Object Tracking Challenge*, pages 191–217, 6-12 Sep 2014.

[74] J. Kwon and K. Lee. Tracking by sampling and integrating multiple trackers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(7):1428–1441, Jul 2014.

[75] D-Y. Lee, J-Y. Sim, and C-S. Kim. Visual tracking using pertinent patch selection and masking. In *IEEE International Conference on Computer Vision and Pattern Recognition*, pages 3486–3493, 23-28 Jun 2014.

[76] I. Leichter, M. Lindenbaum, and E. Rivlin. A general framework for combining visual trackers: the black boxes approach. *International Journal of Computer Vision*, 67:343–363, May 2006.

[77] A. Li, M. Lin, Y. Wu, M.H. Yang, and S. Yan. NUS-PRO: A new visual tracking challenge. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(2):335–349, Feb 2016.

[78] H. Li, Y. Li, and F. Porikli. Deeptrack: Learning discriminative feature representations online for robust visual tracking. *IEEE Transactions on Image Processing*, 25(4):1834–1848, Apr 2016.

[79] Q. Li, X. Wang, W. Wang, Y. Jiang, Z. Zhou, and Z. Tu. Disagreement-based multi-system tracking. In *Proceedings of Asian Conference on Computer Vision*, pages 320–334, 5-9 Nov 2012.

[80] X. Li, W. Hu, Z. Zhang, X. Zhang, and G. Luo. Robust visual tracking based on incremental tensor subspace learning. In *IEEE International Conference on Computer Vision*, pages 1–8, 14-21 Oct 2007.

[81] Y. Li and J. Zhu. A scale adaptive kernel correlation filter tracker with feature integration. In *Proceedings of European Conference on Computer Vision*, pages 254–265, 6-12 Sep 2014.

[82] L. Liu, C. Shen, and A. van den Hengel. The treasure beneath convolutional layers: Cross-convolutional-layer pooling for image classification. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 4749–4757, 7-12 Jun 2015.

[83] R. Liu, S. Li, X. Yuan, and R. He. Online determination of track loss using template inverse matching. In *International Workshop on Visual Surveillance*, pages 17–24, 17 Oct 2008.

[84] S. Liu, T. Zhang, X. Cao, and C. Xu. Structural correlation filter for robust visual tracking. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 4312–4320, 27-30 Jun 2016.

[85] Y. Liu, J. Cui, H. Zhao, and H. Zha. Fusion of low-and high-dimensional approaches by trackers sampling for generic human motion tracking. In *International Conference on Pattern Recognition*, pages 898–901, 11-15 Nov 2012.

[86] J.F. Lucas. *Introduction to Abstract Mathematics*. Rowman and Littlefield, 2 edition, 1990.

[87] C. Ma, J. B. Huang, X. Yang, and M. H. Yang. Hierarchical convolutional features for visual tracking. In *IEEE International Conference on Computer Vision*, pages 3074–3082, 7-13 Dec 2015.

[88] J. Ma, C. Han, and Y. Yang. Visual tracking based on adaptive multi-cue integration. In *International Conference on Information Fusion*, pages 1737–1742, 6-9 Jul 2009.

[89] E. Maggio and A Cavallaro. Hybrid particle filter and mean shift tracker with adaptive transition model. In *IEEE International Conference on Acoustics Speech and Signal Processing*, pages 221–224, 23 Mar 2005.

[90] E. Maggio and A. Cavallaro. *Video Tracking: Theory and Practice*. Wiley, 1st edition, 2011.

[91] E. Maggio, F. Smerladi, and A. Cavallaro. Adaptive multifeature tracking in a particle filtering framework. *IEEE Transactions on Circuits and Systems for Video Technology*, 17(10):1348–1359, Oct 2007.

[92] A. Makris, D. Kosmopoulos, S. Perantonis, and S. Theodoridis. A hierarchical feature fusion framework for adaptive visual tracking. *Image and Vision Computing*, 29(9):594–606, Aug 2011.

[93] X. Mei and H. Ling. Robust visual tracking and vehicle classification via sparse representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(11):2259–2272, Nov 2011.

[94] K. Meshgi, S. Maeda, S. Oba, S. Skibbe, Y. Li, and S. Ishii. An occlusion-aware particle filter tracker to handle complex and persistent occlusions. *Computer Vision and Image Understanding*, 150:81–94, Sep 2016.

[95] D.C. Montgomery, C.L. Jennings, and M. Kulahci. *Introduction to Time Series Analysis and Forecasting*. Wiley Series in Probability and Statistics. Wiley, 2011.

[96] F. Moreno-Noguer, A. Sanfeliu, and D. Samaras. Dependent multiple cue integration for robust tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(4):670–685, Apr 2008.

[97] H. Nam and B. Han. Learning multi-domain convolutional neural networks for visual tracking. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 4293–4302, 27-30 Jun 2016.

[98] T. Nawaz and A. Cavallaro. A protocol for evaluating video trackers under real-world conditions. *IEEE Transactions on Image Processing*, 22(4):1354–1361, Apr 2013.

[99] H.T. Nguyen and A.W.M. Smeulders. Robust tracking using foreground-background texture discrimination. *International Journal of Computer Vision*, 69(3):277–293, 2006.

[100] J. Ning, W. Shi, S. Yang, and P. Yanne. Visual tracking based on distribution fields and online weighted multiple instance learning. *Image and Vision Computing*, 31(11):853–863, Nov 2013.

[101] J. Ning, L. Zhang, D. Zhang, and C. Wu. Robust object tracking using joint color-texture histogram. *International Journal of Pattern Recognition and Artificial Intelligence*, 23(07):1245–1263, Feb 2009.

[102] J. Ning, L. Zhang, D. Zhang, and C. Wu. Scale and orientation adaptive mean shift tracking. *IET Computer Vision*, 6(1):52–61, Jan 2012.

[103] K. Nummiaro, E. Koller-Meier, and L. Van Gool. An adaptive color-based particle filter. *Image and Vision Computing*, 21(1):99–110, Jan 2003.

[104] S. Oron, A. Bar-Hillel, D. Levi, and S. Avidan. Locally orderless tracking. *International Journal Computer Vision*, 111(2):213–228, Jan 2014.

[105] P. Pan, F. Porikli, and D. Schonfeld. A new method for tracking performance evaluation based on a reflective model and perturbation analysis. In *IEEE International Conference on Acoustics Speech and Signal Processing*, pages 3529–3532, 19-24 Apr 2009.

[106] P. Perez, J. Vermaak, and A. Blake. Data fusion for visual tracking with particles. *Proceedings of the IEEE*, 92(3):495–513, Mar 2004.

[107] Y. Qi, S. Zhang, L. Qin, H. Yao, Q. Huang, J. Lim, and M. H. Yang. Hedged deep tracking. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 4303–4311, 27-30 Jun 2016.

[108] X. Ren and J. Malik. Learning a classification model for segmentation. In *IEEE International Conference on Computer Vision*, pages 10–17, 13-16 Oct 2003.

[109] B. Ristic, S. Arulampalam, and N. Gordon. *Beyond the Kalman filter: particle filters for tracking applications*. Artech House, Boston, London, 2004.

[110] D.A. Ross, J. Lim, R. Lin, and M. Yang. Incremental learning for robust visual tracking. *International Journal of Computer Vision*, 77(1):125–141, May 2008.

[111] Y. Rubner, C. Tomasi, and L.J. Guibas. The earth mover's distance as a metric for image retrieval. *International Journal of Computer Vision*, 40(2):99–121, Nov 2000.

[112] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, Dec 2015.

[113] J.C. SanMiguel and A. Calvo. Covariance-based online validation of video tracking. *IEEE Electronics Letters*, 51(3):226–228, Feb 2015.

[114] J.C. SanMiguel and A. Cavallaro. Temporal validation of particle filters for video tracking. *Computer Vision and Image Understanding*, 131(1):42–55, Feb 2015.

[115] J.C. SanMiguel, A. Cavallaro, and J.M. Martinez. Evaluation of on-line quality estimators for object tracking. In *IEEE International Conference on Image Processing*, pages 825–828, 26-29 Sep 2010.

[116] J.C. SanMiguel, A. Cavallaro, and J.M. Martinez. Adaptive online performance evaluation of video trackers. *IEEE Transactions on Image Processing*, 21(5):2812–2823, May 2012.

[117] J.C. SanMiguel, A. Cavallaro, and J.M. Martinez. Standalone evaluation of deterministic video tracking. In *IEEE International Conference on Image Processing*, pages 1353–1356, 30 Sep-3 Oct 2012.

[118] J. Santner, C. Leistner, A. Saffari, T. Pock, and H. Bischof. PROST: Parallel robust online simple tracking. In *IEEE International Conference on Computer Vision and Pattern Recognition*, pages 723–730, 13-18 Jun 2010.

[119] N.I. Sapankevych and R. Sankar. Time series prediction using support vector machines: A survey. *IEEE Computational Intelligence Magazine*, 4(2):24–38, May 2009.

[120] T. Schlogl, C. Beleznai, M. Winter, and H. Bischof. Performance evaluation metrics for motion detection and tracking. In *International Conference on Pattern Recognition*, pages 519–522, 23-26 Aug 2004.

[121] T. Senst, V. Eiselein, and T. Sikora. Robust local optical flow for feature tracking. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(9):1377–1387, Sep 2012.

[122] L. Sevilla-Lara and E. Learned-Miller. Distribution fields for tracking. In *IEEE International Conference on Computer Vision and Pattern Recognition*, pages 1910–1917, 16-21 Jun 2012.

[123] K. Shearer, K.D. Wong, and S. Venkatesh. Combining multiple tracking algorithms for improved general performance. *Pattern Recognition*, 34(6):1257–1269, Jun 2001.

[124] N. Siebel and S. Maybank. Fusion of multiple tracking algorithms for robust people tracking. In *Proceedings of European Conference on Computer Vision*, pages 1–15, 28-31 May 2002.

[125] A. Smeulders, D. Chu, R. Cucchiara, S. Calderara, A. Dehghan, and M. Shah. Visual tracking: An experimental survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(7):1442–1468, Jul 2014.

[126] C. Spampinato, S. Palazzo, and D. Giordano. Evaluation of tracking algorithm performance without ground-truth data. In *IEEE International Conference on Image Processing*, pages 1345–1348, 30 Sep-3 Oct 2012.

[127] M. Spengler and B. Schiele. Towards robust multi-cue integration for visual tracking. *Machine Vision and Applications*, 14(1):50–58, Apr 2003.

[128] C. Stauffer and W.E.L. Grimson. Learning patterns of activity using real-time tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):747–757, Aug 2000.

[129] B. Stenger, T. Woodley, and R. Cipolla. Learning to track with multiple observers. In *IEEE International Conference on Computer Vision and Pattern Recognition*, pages 2647–2654, 20-28 Jun 2009.

[130] M. Tang and J. Feng. Multi-kernel correlation filter for visual tracking. In *IEEE International Conference on Computer Vision*, pages 3038–3046, 7-13 Dec 2015.

[131] J.W. Taylor and P.E. McSharry. Short-term load forecasting methods: An evaluation based on european data. *IEEE Transactions on Power Systems*, 22(4):2213–2219, Nov 2007.

[132] M. Tian, W. Zhang, and F. Liu. On-line ensemble svm for robust object tracking. In *Asian Conference on Computer Vision*, pages 355–364, 18-22 Nov 2007.

[133] J. Triesch and C.V.D Malsburg. Democratic integration: Self-organized integration of adaptive cues. *Neural Computing*, 13(9):2049–2074, Sep 2001.

[134] N. Vaswani. Additive change detection in nonlinear systems with unknown change parameters. *IEEE Transactions on Signal Processing*, 55(3):859–872, Mar 2007.

[135] H. Veeraraghavan, P. Schrater, and N. Papanikolopoulos. Robust target detection and tracking through integration of motion, color, and geometry. *Computer Vision and Image Understanding*, 103(2):121–138, Aug 2006.

[136] H.D. Vinod and P. Basu. Forecasting consumption, income and real interest rates from alternative state space models. *International Journal of Forecasting*, 11(2):217–231, Jun 1995.

[137] P. Viola, J. C. Platt, and C. Zhang. Multiple instance boosting for object detection. In *Proceedings of Neural Information Processing Systems*, pages 1419–1426, 4-9 Dec 2006.

[138] T. Vojir and J. Matas. The enhanced flock of trackers. In *Registration and Recognition in Images and Videos*, pages 113–136. Springer Berlin Heidelberg, 2014.

[139] T. Vojir, J. Matas, and J. Noskova. Online adaptive hidden markov model for multi-tracker fusion. *Computer Vision and Image Understanding*, 2016.

[140] T. Vojir, J. Noskova, and J. Matas. Robust scale-adaptive mean-shift for tracking. *Pattern Recognition Letters*, 49:250–258, Nov 2014.

[141] G.S. Walia and R. Kapoor. Recent advances on multicue object tracking: a survey. *Artificial Intelligence Review*, 46(1):1–39, 2016.

[142] D. Wang, H. Lu, and M. Yang. Least soft-threshold squares tracking. In *IEEE International Conference on Computer Vision and Pattern Recognition*, pages 2371–2378, 23-28 Jun 2013.

[143] D. Wang, H. Lu, and M-H. Yang. Online object tracking with sparse prototypes. *IEEE Transactions on Image Processing*, 22(1):314–325, Jan 2013.

[144] L. Wang, W. Ouyang, X. Wang, and H. Lu. Visual tracking with fully convolutional networks. In *IEEE International Conference on Computer Vision*, pages 3119–3127, 7-13 Dec 2015.

[145] L. Wang, W. Ouyang, X. Wang, and H. Lu. STCT: Sequentially training convolutional networks for visual tracking. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1373–1381, 27-30 Jun 2016.

[146] Y. Wang, H. Chen, S. Li, J. Zhang, and C. Gao. Object tracking by color distribution fields with adaptive hierarchical structure. *The Visual Computer*, pages 1–13, Nov 2015.

[147] M. Wasif and J.F. Schmidt. Autoregressive integrated model for time synchronization in wireless sensor networks. In *ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pages 133–140, 2-6 Nov 2015.

[148] J. Whitehill, P. Ruvolo, T. Wu, J. Bergsma, and J. Movellan. Whose vote should count more: Optimal integration of labels from labelers of unknown expertise. In *International Conference on Neural Information Processing Systems*, pages 2035–2043, 5-8 Dec 2009.

[149] H. Wu, A.C. Sankaranarayanan, and R. Chellappa. Online empirical evaluation of tracking algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(8):1443–1458, Aug 2010.

[150] H. Wu and Q. Zheng. Self-evaluation for video tracking systems. In *Proceedings for the Army Science Conference*, 29 Nov-2 Dec 2004.

[151] J. Wu, S. Hu, and Y. Wang. Adaptive multi-feature visual tracking in a probability-hypothesis-density filtering framework. *Signal Processing*, 93(11):2915–2926, Nov 2013.

[152] Y. Wu and T.S. Huang. A co-inference approach to robust visual tracking. In *IEEE International Conference on Computer Vision*, pages 26–33, 7-14 Jul 2001.

[153] Y. Wu and T.S. Huang. Robust visual tracking by integrating multiple cues based on co-inference learning. *International Journal of Computer Vision*, 58:55–71, Nov 2002.

[154] Y. Wu, J. Lim, and M. Yang. Object tracking benchmark. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(9):1834–1848, Sep 2015.

[155] Y. Wu, J. Lim, and M-H. Yang. Online object tracking: A benchmark. In *IEEE International Conference on Computer Vision and Pattern Recognition*, pages 2411–2418, 23-28 Jun 2013.

[156] H. Yang, L Shao, F. Zheng, L. Wang, and Z Song. Recent advances and trends in visual tracking: A review. *Neurocomputing*, 74(18):3823–3831, Nov 2011.

[157] M. Yang, Y. Wu, and G. Hua. Context-aware visual tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(7):1195–1209, Jul 2009.

[158] A. Yilmaz, O. Javed, and M. Shah. Object tracking: A survey. *ACM Computing Surveys*, 38(4), Dec 2006.

[159] J. H. Yoon, M. H. Yang, and K.J. Yoon. Interacting multiview tracker. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(5):903–917, May 2016.

[160] J.H. Yoon, D.Y. Kim, and K.J. Yoon. Visual tracking via adaptive tracker selection with multiple features. In *Proceedings of European Conference on Computer Vision*, pages 28–41, 7-13 Oct 2012.

[161] Z. Yu, L. Li, J. Liu, and G. Han. Hybrid adaptive classifier ensemble. *IEEE Transactions on Cybernetics*, 45(2):177–190, Feb 2015.

[162] Y. Yuan, S. Emmanuel, W. Lin, and Y. Fang. Visual object tracking based on appearance model selection. In *IEEE International Conference on Multimedia and Expo Workshops*, pages 1–4, 15-19 Jul 2013.

[163] Y. Zhai, M.B. Yeary, S. Cheng, and N. Kehtarnavaz. An object-tracking algorithm based on multiple-model particle filtering with state partitioning. *IEEE Transactions on Instrumentation and Measurement*, 58(5):1797–1809, May 2009.

[164] K. Zhang, Q. Liu, Y. Wu, and M. H. Yang. Robust visual tracking via convolutional

networks without training. *IEEE Transactions on Image Processing*, 25(4):1779–1792, Apr 2016.

[165] K. Zhang, L. Zhang, and M-H. Yang. Real-time compressive tracking. In *Proceedings of European Conference on Computer Vision*, pages 864–877, 7-13 Oct 2012.

[166] K. Zhang, L. Zhang, and M-H.. Yang. Fast compressive tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(10):2002–2015, Oct 2014.

[167] S. Zhang, H. Yao, X. Sun, and X. Lu. Sparse coding based visual tracking: Review and experimental comparison. *Pattern Recognition*, 46(7):1772 – 1788, Jul 2013.

[168] X. Zhang, W. Hu, H. Bao, and S. Maybank. Robust head tracking based on multiple cues fusion in the kernel-bayesian framework. *IEEE Transactions on Circuits and Systems for Video Technology*, 23(7):1197–1208, Jul 2013.

[169] B. Zhong, H. Yao, S. Chen, R. Ji, T. Chin, and H. Wang. Visual tracking via weakly supervised learning from multiple imperfect oracles. *Pattern Recognition*, 47(3):1395–1410, Mar 2014.

[170] B. Zhong, H. Yao, S. Chen, R. Ji, X. Yuan, S. Liu, and W. Gao. Visual tracking via weakly supervised learning from multiple imperfect oracles. In *IEEE International Conference on Computer Vision and Pattern Recognition*, pages 1323–1330, 13-18 Jun 2010.

[171] S.K. Zhou, R. Chellappa, and B. Moghaddam. Visual tracking and recognition using appearance-adaptive models in particle filters. *IEEE Transactions on Image Processing*, 13(11):1491–1506, Nov 2004.