Improving the Design and Use of Correlation Filters in Visual Tracking

Submitted in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy
in
Electrical and Computer Engineering

Stephen A. Siena

B.S., Computer Science, University of Notre Dame
M.S., Electrical and Computer Engineering, Carnegie Mellon University

Carnegie Mellon University
Pittsburgh, PA

September, 2017

Thesis Committee Members

Prof. Vijayakumar Bhagavatula (Advisor)
Department of Electrical and Computer Engineering
Carnegie Mellon University

Prof. Marios Savvides
Department of Electrical and Computer Engineering
Carnegie Mellon University

Prof. Simon Lucey
Robotics Institute
Carnegie Mellon University

Dr. Abhijit Mahalanobis
Technical Fellow
Lockheed Martin Corporation

Abstract

The volume of video data collected will only increase as the prevalence of automated systems continues to grow and those systems start to rely more on vision sensors to make decisions. Visual tracking is the process of automatically estimating the location of an object through the course of a video. The ability to track objects in video is useful in applications such as autonomous driving, surveillance, and robotics. The ability to track objects allows for more effective decision making processes for tasks such as predictive driving, anomaly detection, and face recognition. With the amount of data to parse and the benefits of doing so accurately, the need for fast and reliable visual tracking is clear.

Correlation filters, previously used in detection and recognition tasks within single images, have become a popular approach to visual tracking because of their ability to efficiently match and align two images. Correlation filters have been adapted for visual tracking by developing incremental learning techniques, allowing efficient updating of correlation filters. Tracker elements such as more powerful feature representations and improved scale tolerance have led to state-of-the-art tracking performance.

Still, despite the recent improvements in correlation filter trackers, there remain unexplored aspects of the union of correlation filters and visual tracking. This work explores alternative correlation filter designs that have not previously been adapted to visual tracking. We also introduce an occlusion detection system to address situations where the targets are temporarily not visible; one of the most challenging aspects of tracking. We validate our approaches on widely used benchmarks while also introducing a new evaluation metric that reflects the amount of activity that occurs within a given video.

Acknowledgments

I want to first thank my advisor, Professor Vijayakumar Bhagavatula. His dedication to the success of the students he advises is beyond what anyone could ask of him. It is a blessing to have worked with someone so knowledgeable, accessible, and patient.

I would like to thank my committee members, Professor Marios Savvides, Professor Simon Lucey, and Dr. Abhijit Mahalanobis for their interest in my thesis work and discussions to improve it. I would also like to thank the CyLab Usable Privacy and Security Laboratory (CUPS), the Air Force Research Laboratory (AFRL), and David Barakat and LaVerne Owen-Barakat for their support throughout my education.

More members of the ECE community than I can list helped me immensely throughout my time at CMU. I thank Andres Rodriguez, Vishnu Naresh Boddeti, Joey Fernandez, Kathy Brigham, and Jon Smereka for welcoming me into the ECE community and setting examples through the high standards they always displayed. I also thank Yongjune Kim, Zhiding Yu, Andrew Fox, Eric He, and Rick Chang for their helpful feedback from countless discussions over the years. I thank Marilyn Patete and Kara Knickerbocker for making the B200 Wing a community and not just an office space. Finally, I thank Dave Fortna, Leona Kass, Katie Costa, and the officers of the ECE Graduate Organization for their friendship and willingness to help improve the entire ECE graduate student community over the years.

Completing this thesis would not have been possible without the support of friends and family. Thanks to the Gigahurtz softball team, Fast Floorier Transforms floor hockey team, and all my friends within the ECE department and the CMU community. Thanks to Rob, John, Connor, Patrick, Michael, Brian, and Doug for their friendly support over the years that kept me grounded even in the most challenging times.

Finally and most of all, I want to thank my parents and my brother and sister, whose love and overwhelming desire to see me succeed meant more than could be put into words.

Acronyms

ACE average correlation energy
ACT Adaptive Color Tracker
ASO average segment overlap
ATR automatic target recognition
AVO average video overlap
CCT Collaborative Correlation Tracker
CF correlation filter
CFC correlation filter coding
CFT correlation filter tracker
CLE central location error
CNN convolutional neural network
CSK Circulant Structure Kernel
DFT discrete Fourier transform
DSST Discriminative Scale Space Tracker
EBT Edge Box Tracker
ECPSDF Equal Correlation Peak Synthetic Discriminant Function
FFT fast Fourier transform
FPS frames per second
GGT Geometric Hypergraph Tracker
HOG histogram of oriented gradients
HuBOE Hue-Based Occlusion Estimation
IDFT inverse discrete Fourier transform
KCF Kernelized Correlation Filter
MACE Minimum Average Correlation Energy
MAD Median Absolute Deviations
MDNet Multi-Domain Network
MF matched filter

MKCF Multi-Kernelized Correlation Filter

MLDF Multi-Level Deep Feature Tracker

MMCF Maxmimum Margin Correlation Filter

MOSSE minimum output sum of squared error

MSE mean squared error

MUSTer MUlti-Store Tracker

MvCFT Multi-view Correlation Filter Tracker

MVSDF Minimum Variance Synthetic Discriminant Function

NN neural network

ONV output noise variance

OTB Online Tracking Benchmark

OTSDF Optimal Tradeoff Synthetic Discriminant Function

PCE peak to correlation energy

PSD power spectral density

PSR peak-sidelobe ratio

SAMF Scale Adaptive with Multiple Features

SCM Sparsity-based Collaborative Model

SIFT Scale-Invariant Feature Transform

SNR signal-to-noise ratio

SRBT Salient Region Based Tracker

SRDCF Spatially Regularized Discriminative Correlation Filters

Staple Sum of Template And Pixel-wise LEarners

SVM support vector machine

TCNN Tree-Structured Convolutional Neural Network

TLD Tracking-Learning-Detection

UOTSDF Unconstrained Optimal Tradeoff Synthetic Discriminant Function

VOT Visual Object Tracking

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Visual tracking is an important capability for applications including robotics, surveillance, and sports video analysis. Visual tracking – the task of following an object during the course of a video – is often a critical intermediate step scene analysis, e.g., identifying anomalous behavior in surveillance videos or processing images sequences from robots operating autonomously in dynamic environments. Tracking arbitrary objects in unconstrained environments can be a challenging task that requires tolerance to a range of factors including rotations, deformations, and scale changes of the target object as well as occlusions, clutter, and lighting changes caused by the object's surroundings. When there is minimal information about the target object, these factors can occur individually or together at unpredictable times.

In this thesis, we focus primarily on *single-camera*, *single-target*, *model-free*, *short-term*, *causal* tracking. *Single-target*, *single-camera* tracking means there is only one object of interest being tracked at a time, and only one sensor capturing the object during the duration of the test period. *Model-free* tracking indicates that at no point during tracking is the object classified (e.g., car, person, ball) in order to leverage a pre-trained model specifically to detect and track that particular type of object. The threshold for what constitutes *short-term* tracking may not be precise, but the length of the videos considered

in this work are on the order of seconds or a couple of minutes, and the disappearance and reappearance (and redetection) of objects is not a concern, nor is there a concern that objects will fundamentally change their appearance (e.g., a person changing outfits). Finally, *causal* tracking indicates that the estimated target location at time t_k will depend solely on information from earlier video information where time $t \leq t_k$, rather than refining track estimates using information from parts of the video where time $t > t_k$. Additionally, the trackers will only have a single initialization frame to learn the appearance of the target object. This condition, along with the model-free restriction, makes this a very difficult setting for successful tracking, and is reasonably a lower bound for well-defined tracking scenarios where the target object – possibly in terms of the object's classification, amount of prior appearance data, or knowledge of the environment – is much better understood. A tracking system that performs well under the restrictions we will work under can either indicate a good starting point to building a more specialized tracker, or can itself be a desirable tracker choice to be deployed in varied or unknown operating conditions.

In recent years, the focus of visual tracking research has shifted more to *tracking-by-detection*. Tracking-by-detection methods typically rely heavily on an object detector similar to what may be found in single-image object detection problems in other computer vision applications. The strong discriminative power of newer object detectors and feature representations diminishes the need for applying more sophisticated motion models to tracking objects. Instead, a robust object detector is often just applied in the local region of the previous target location, and then the object detector may be subsequently updated with data from the changing target; the motion of the object in recent frames is often not used when finding the target location in the next video frame.

The increased use of object detectors, along with the desire to process videos efficiently (and, in some applications, in real time), make the use of correlation filters (CFs) appealing in a system that performs tracking-by-detection. CFs have previously been effectively used

in applications such as automatic target recognition (ATR), object alignment, and biometric recognition, and their efficient computation in the frequency domain lends itself naturally to the visual tracking problem. Accordingly, the number of correlation filter trackers (CFTs) has consistently grown in recent years and ways to improve both the accuracy and speed of CFTs is an active area of research.

In the remainder of this chapter we provide overviews of the visual tracking problem and CFs, and outline the main contributions of this thesis. We also introduce notation used in this thesis and outline the structure of the rest of the thesis.

1.1 Technical Background

1.1.1 Visual Tracking

This thesis focuses on improving performance in visual tracking. In online visual tracking, we are given a tracker, denoted $\mathcal{T}$, and a video[1] denoted $\mathcal{I}$, comprised of $n+1$ frames[2] such that $\mathcal{I} = \{I_0, I_1, \ldots, I_n\}$. For the kth image I_k, there is a rectangular region $\mathbf{b}_k = [x_k, y_k, w_k, h_k]$ marked by a human reader, where $[x_k, y_k]$ represents the center location of the target and $[w_k, h_k]$ represents the width and height of the target. The use of a rectangular bounding box means that there is often a small number of pixels inside the box that actually correspond to the background or surroundings. The tracker is only provided the initial bounding box $\mathbf{b}_0 = [x_0, y_0, w_0, h_0]$ to learn the appearance of the target, although many tracker systems use data from subsequent frames using their own target location estimates, given the considerable difficulties of building a robust target model from a single image.

The tracking system outputs estimates $\mathbf{b}'_k = [x'_k, y'_k, w'_k, h'_k]$ for the n frames remaining in the sequence. There are a number of different ways to quantify the performance of a tracker, and the approaches used in this thesis are outlined in Sec. 3.1.2.

[1]The terms "video," "sequence," and "image sequence" are used interchangeably.
[2]The terms "frames" and "images" are used interchangeably.

1.1.2 Correlation Filters

CFs have been used in computer vision applications including vehicle recognition, object alignment, and biometric localization and recognition. Different designs of CFs can be used to be tolerant to target noise and distortions. Additionally, most CFs exhibit graceful degradation and have built-in shift invariance. All together, these properties make CFs suitable for simultaneous localization and recognition, which is what successful visual tracking demands.

In CFTs, a template $h(m,n)$[3] is cross-correlated with a new video frame (or subregion of a frame) $x(m,n)$ of a tracking video, producing a correlation output $g(m,n)$. This cross-correlation is computed efficiently via the frequency domain:

$$G(u,v) = X(u,v)H^*(u,v) \tag{1.1}$$

where H^* is the complex conjugate of H and $G(u,v)$, $X(u,v)$, and $H(u,v)$ are the 2D discrete Fourier transforms (DFTs) of $g(m,n)$, $x(m,n)$, and $h(m,n)$, respectively. To analyze the correlation output, we compute the inverse discrete Fourier transform (IDFT) of the output $G(u,v)$. All the required 2D DFTs are usually computed using the efficient fast Fourier transform (FFT) algorithms. In ideal circumstances, there will be a sharp peak or high correlation in the correlation plane at the location of the detected object and low correlation elsewhere in the output, as shown in Fig. 1.1.[4] In most visual tracking applications, the target's location is estimated as the location of the highest correlation value in the plane.

[3] "Templates" refer to space domain representations while "filters" refer to their frequency domain counterparts.

[4] In a recognition setting with a centered query image, we may expect a sharp centered peak for a true-class image and no such sharp peak for other images.

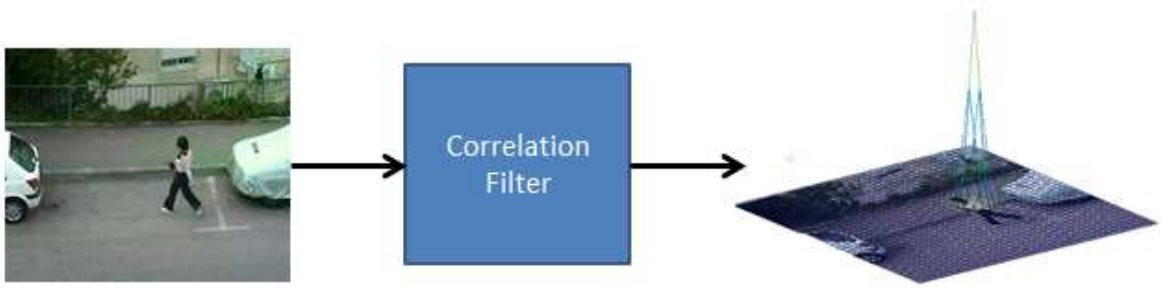


Figure 1.1: Correlation filters will ideally produce a sharp peak at the center location of the target it is designed to locate, with low correlation elsewhere in the correlation plane.

1.2 Thesis Contributions

This thesis' primary focus is on improving our understanding of and the performance of CFTs. This thesis presents the following contributions.

1.2.1 An Evaluation Tracking Metric to Reflect Qualitative Observation

There are a number of measures to quantify the performance of a tracker within a single frame, a full track, and an entire dataset. While all measures are precisely defined from a mathematical perspective, they often do not entirely align with how a human observer would evaluate a single tracker or compare different trackers. This is challenging due to the fact that consecutive frames are often highly related and due to the fact that the challenging elements within a video or within multiple videos occur at irregular intervals. We propose a new accuracy metric designed to address these elements by segmenting videos according to the amount of activity that occurs within it, and then computing a metric according to those segments. A statistical test between different trackers can also be performed to provide further insight beyond just the raw difference between trackers, which is often the only option with other measures.

1.2.2 Correlation Filter Design for Tracking

There has been a dramatic increase in the number of CFTs introduced in recent years, and there is a wealth of research leading to different CF designs. Despite this, there appears to be very little intersection between these two topics of research, with most CFTs being based on one core CF design. We evaluate the use of alternative CF designs in a fundamental CFT framework. We substitute different CFs in a tracker, and then further introduce modified training schemes that are tailored to the introduced CF designs for the tracking problem. We also discuss the fusion of multiple CFs for improved tracking accuracy.

1.2.3 Occlusion Detection for Model-Free Tracking in Color Video

The challenge of model-free tracking is the inability to leverage any external training data that may give the tracker knowledge about how a target object may appear during the course of tracking. Instead, the tracker must adapt to previously unseen changes in the target appearance. These changes can be small and gradual, or very sudden and more dramatic. Trackers are thus required to consistently update their detection model, but this also makes trackers susceptible to confusing occlusions for large appearance changes. While most trackers simply balance their adaptability with their robustness to occlusions, we introduce a method based on color information that allows a tracker to automatically change its adaptation rate depending on if the target is estimated to be occluded or not. By reducing the rate at which a tracker updates its target model when the target is occluded, we prevent the tracker from learning the appearance of the occluding object and retain the appearance information of the true target. We show that this approach works effectively and improves the performance of a number of CFTs.

1.3 Notation

- lower case bold letters indicate column vectors, e.g., $\mathbf{x}$
- upper case bold letters indicate matrices, e.g., $\mathbf{X} = [\mathbf{x}_1, ..., \mathbf{x}_n]$
- some matrices are diagonal matrix representations of vectors, e.g., $\mathbf{X}_i = \text{diag}(\mathbf{x}_i)$
- hat symbol indicates frequency-domain variables, e.g., $\hat{\mathbf{x}}$
- the superscript T symbol denotes the transpose, e.g., $\mathbf{x}^T$
- the superscript asterisk symbol denotes the complex conjugate, e.g., $\hat{\mathbf{x}}^*$
- the superscript dagger symbol denotes the complex conjugate transpose, e.g., $\hat{\mathbf{x}}^\dagger$

1.4 Thesis Organization

The rest of this thesis is organized as follows: Chapter 2 gives an overview of CFs, early CFTs, and discusses advances in a number of specific tracking system elements. Chapter 3 discusses how these trackers are evaluated, and proposes a new accuracy metric to characterize performance. Chapter 4 discusses further the CF design decisions in various CFTs and explores the use of CFs not previously applied to visual trackers. Chapter 5 discusses a technique for detecting occlusion in videos to improve the performance of CFTs, independent of many various tracker design decisions. Finally, Chapter 6 summarizes the contributions of this thesis and proposes future avenues of continued research.

Chapter 2

Correlation Filter Trackers

2.1 Introduction

We review advances in CFs and CFTs in this chapter. We start by discussing CFs, beginning with the most fundamental designs and showing the progression to the most commonly used and recent CF designs. While most early CFs were not designed for visual tracking applications, their design principles influenced the subsequent CF designs that were tailored for visual tracking. Additionally, we believe some CF designs deserve a deeper exploration for their application in visual tracking; for those filters, the material in this chapter is an introduction to work discussed in Ch. 4.

Following a discussion of CF design in Sec. 2.2, we introduce the minimum output sum of squared error (MOSSE) filter, which was one of the first CFs designed for the task of visual tracking, and the core of the most influential early CFT (generally referred to as the "MOSSE tracker") [1]. Following our discussion of the MOSSE filter, which was introduced in 2010, we discuss the Kernelized Correlation Filter (KCF) tracker [2], which was originally introduced in 2012 as the Circulant Structure Kernel (CSK) tracker [3]. These two formulations showed greatly improved results compared to the original MOSSE filter, as well as other non-CF trackers at the time. The fundamental CF design in most

subsequent CFTs are heavily adopted from either the MOSSE or KCF trackers, which primarily introduced novel CF designs in otherwise simple trackers.

Since the introduction of the KCF tracker, a number of CFTs have been introduced. Although some CFTs introduce small changes to the filter design, most focus on particular aspects of the visual tracking problems posed in benchmark datasets and in real-world situations. These particular elements include introducing methods to handle occlusion, adapting CFs to estimating a target's scale, and incorporating redetection schemes. Other CFTs have proposed using different feature extractors; different features do not address a particular problem, but are expected to improve tracking across a range of challenges. We will discuss this variety of improvements while highlighting only the novel parts of different CFTs.

We conclude the chapter with a discussion of non-CF trackers. There are trackers that predate the MOSSE or KCF trackers as well as non-CF trackers that have been introduced since their publication. We discuss these other trackers to provide additional context to the body of work in CFTs, and to contrast the strengths and weaknesses of eschewing the CF approach to tracking.

2.2 Introduction to Correlation Filters

This section will introduce CFs beyond the MOSSE filter introduced in Sec. 2.3.1. Along with differences in filter design, it is important to emphasize that the filters introduced within this section were not designed with the goal of visual tracking. Instead, single-image localization and recognition tasks were generally the objective of these filters.

Once any CF is computed, the application of it to process a test image almost always follows the same procedure. Given the learned or designed correlation template $\mathbf{h}$ and a test image $\mathbf{x}_{\text{test}}$, we can efficiently compute the correlation via the Fourier domain

$$
\begin{aligned}
\mathbf{g} &= \mathbf{x}_{\text{test}} \otimes \mathbf{h} \\
&= \mathcal{F}^{-1}\left\{\mathcal{F}\{\mathbf{x}_{\text{test}}\} \odot \mathcal{F}^{*}\{\mathbf{h}\}\right\},
\end{aligned}
\tag{2.1}
$$

where $\mathbf{g}$ is the correlation output, $\otimes$ represents the cross-correlation operation, $\mathcal{F}$ and $\mathcal{F}^{-1}$ represent the DFT and IDFT operators, respectively, the $*$ superscript represents the conjugate, and $\odot$ represents the Hadamard product. To avoid circular convolution, $\mathbf{x}$ and $\mathbf{h}$ are zero-padded prior to the DFT operation. When analyzing $\mathbf{g}$, the straightforward approach is to estimate a target location as the location of the highest value in $\mathbf{g}$. This is a viable option when localizing a target instead of detecting it, i.e., estimating a target's location instead of determining if a target is present or not. Detecting a target requires setting a threshold on the correlation value above which the presence of a target is declared. Setting a threshold on the correlation values is possible, or the correlation plane can be processed to find the peak-sidelobe ratio (PSR) values instead. PSR is computed as

$$
\text{PSR} \triangleq \frac{\text{peak} - \mu}{\sigma}, \tag{2.2}
$$

where μ and σ are the mean and standard deviations of the correlation values in the sidelobe region of a correlation peak. The sidelobe region is defined as a toroidal region surrounding the peak that excludes the peak itself and possibly a small region closest to the peak. PSR can normalize the quality of a match in settings where the total energy in the correlation plane may vary among different images or between different regions of the same image due to illumination variation. The PSR can be calculated selectively at specific peak locations, or for the entire correlation output using two correlations (or four DFTs) [4].

The simplest CF is the matched filter (MF). The MF maximizes the signal-to-noise ratio (SNR) in the presence of additive white noise [5]. In general, the MF does not perform well in scenarios where the test image has a different appearance compared to the training images. Despite this, the MF illustrates and leads into the application of CFs more broadly.

The objective of the MF is to find a filter $\hat{\mathbf{h}}$ that will match well with signal $\hat{\mathbf{x}}$. Recall

that $\hat{\mathbf{h}}$ and $\hat{\mathbf{x}}$[1] are the DFTs [2] of $\mathbf{h}$ and $\mathbf{x}$, respectively. We aim to minimize the mean squared error (MSE), denoted as ϵ, between $\hat{\mathbf{h}}$ and $\hat{\mathbf{x}}$,

$$\begin{aligned}\epsilon &= |\hat{\mathbf{h}} - \hat{\mathbf{x}}|^2 \\ &= (\hat{\mathbf{h}} - \hat{\mathbf{x}})^\dagger(\hat{\mathbf{h}} - \hat{\mathbf{x}}) \\ &= \hat{\mathbf{h}}^\dagger\hat{\mathbf{h}} + \hat{\mathbf{x}}^\dagger\hat{\mathbf{x}} - 2\hat{\mathbf{h}}^\dagger\hat{\mathbf{x}},\end{aligned} \tag{2.3}$$

where † represents the conjugate transpose. If we normalize the energy $\mathbf{h}$ and $\mathbf{x}$ to 1, then $\hat{\mathbf{h}}^\dagger\hat{\mathbf{h}} = 1$ and $\hat{\mathbf{x}}^\dagger\hat{\mathbf{x}} = 1$, and minimizing the error ϵ is equivalent to maximizing $\hat{\mathbf{h}}^\dagger\hat{\mathbf{x}}$, which is accomplished by setting $\hat{\mathbf{h}} = \hat{\mathbf{x}}$.

While the MF is designed to match the training image effectively, in practice the MF performs poorly in recognition scenarios where the target will have appearance differences (loosely called distortions) not seen in training. Effective recognition with CFs requires designing filters with more tolerance to unseen distortions. Additionally, the MF is not designed to produce a sharp peak to detect the target. A sharp correlation peak will localize the target with more precision, and test images that differ from the single training image will not produce strong correlation or a distinct peak in the output correlation plane.

The Equal Correlation Peak Synthetic Discriminant Function (ECPSDF) filter, introduced in 1980 by Hester and Casasent [6], extends the MF by building a composite filter from a larger set of training images. While the MF is built from a single training image, the ECPSDF is built with training signals that reflect a range of possible distortions.

The ECPSDF filter introduces the notion of peak constraints which require the learned filter $\hat{\mathbf{h}}$ to produce certain correlation output values at the origin when correlated with the training images. Assuming we have a training set $\hat{\mathbf{X}} = [\hat{\mathbf{x}}_1, \hat{\mathbf{x}}_2, \ldots, \hat{\mathbf{x}}_n]$, we require that $\hat{\mathbf{h}}^\dagger\hat{\mathbf{x}}_i = u_i$, where $\hat{\mathbf{h}}$ is the filter, $\hat{\mathbf{x}}_i$ represents the ith training image and u_i is the corresponding peak value constraint for the ith image. Typically $u_i = 1$ when x_i is a

[1] In general, $\hat{\mathbf{z}}$ will denote the frequency-domain representation of $\mathbf{z}$.

[2] The DFTs require that $\mathbf{h}$ and $\mathbf{x}$ be zero-padded; if $\mathbf{h}$ and $\mathbf{x}$ are 1D signals of length N_h and N_x, respectively, then both must be zero-padded to at least size $N_h + N_x - 1$. For multidimensional signals, each dimension must each be zero-padded in a similar manner.

true-class image. The other constraint is that the filter $\hat{\mathbf{h}}$ be a linear combination of the training images, i.e., $\hat{\mathbf{h}} = \sum_{i=1}^{n} a_i \hat{\mathbf{x}}_i$. These constraints can be written concisely as

$$\hat{\mathbf{X}}^{\dagger}\hat{\mathbf{h}} = \mathbf{u} \tag{2.4}$$

and

$$\hat{\mathbf{h}} = \hat{\mathbf{X}}\mathbf{a} \tag{2.5}$$

where $\mathbf{u} = [u_1, u_2, \ldots, u_n]^T$ and $\mathbf{a} = [a_1, a_2, \ldots, a_n]^T$. Substitution Eq. 2.5 for $\hat{\mathbf{h}}$ in Eq. 2.4 produces

$$\hat{\mathbf{X}}^{\dagger}\hat{\mathbf{X}}\mathbf{a} = \mathbf{u} \tag{2.6}$$

and, assuming that $\hat{\mathbf{X}}^{\dagger}\hat{\mathbf{X}}$ is nonsingular, we can then solve for $\mathbf{a}$ to obtain

$$\mathbf{a} = (\hat{\mathbf{X}}^{\dagger}\hat{\mathbf{X}})^{-1}\mathbf{u}. \tag{2.7}$$

Eq. 2.7 gives us the weights to solve for the filter $\hat{\mathbf{h}}$, where by substituting Eq. 2.7 into Eq. 2.5 we get the following expression for the filter vector,

$$\hat{\mathbf{h}} = \hat{\mathbf{X}}(\hat{\mathbf{X}}^{\dagger}\hat{\mathbf{X}})^{-1}\mathbf{u}. \tag{2.8}$$

While the ECPSDF filter allows for a single filter to be trained on multiple distortions, it has limitations. While the composite filter can incorporate multiple distortions in its design, it is not robust to unseen distortions, and will not produce sharp correlation output peaks. Just as with the MF, these issues make distinguishing a true-class target from its background very challenging. Subsequent filter designs addressed these concerns.

2.2.1 Minimum Variance Synthetic Discriminant Function Filter

The Minimum Variance Synthetic Discriminant Function (MVSDF) filter was introduced in 1986 by Kumar [7]. The MVSDF filter minimizes the correlation output noise variance (ONV) of a given training set of images in the presence of zero-mean, additive, stationary noise. When the noise is assumed to be white noise, then the MVSDF filter is equal to the ECPSDF filter [6].

In the space domain, we can represent a training sample as $\mathbf{x}_i + \mathbf{n}$, where $\mathbf{x}_i$ is the noiseless training sample and $\mathbf{n}$ is the additive noise. In this scenario, the correlation peak (at the origin) will be $\mathbf{h}^T(\mathbf{x}_i + \mathbf{n}) = c_i + c_n$, where c_i is the desired correlation peak and c_n is the contribution to the peak due to the noise. The ONV is then defined as

$$\begin{aligned} var(c_n) &= E\{(\mathbf{h}^T\mathbf{n})^2\} \\ &= \mathbf{h}^T E\{\mathbf{n}\mathbf{n}^T\}\mathbf{h} \\ &= \mathbf{h}^T\mathbf{C}\mathbf{h} \end{aligned} \tag{2.9}$$

where $\mathbf{C} = E\{\mathbf{n}\mathbf{n}^T\}$ is the $d \times d$ covariance matrix of the noise, where d is the dimension of the training sample.

We can produce a similar solution in the frequency domain as well. When the noise is stationary, we can express $var(c_n)$ by summing over all the frequencies of the the power spectral density (PSD) of $var(c_n)$, denoted S_c. S_c can be expressed as the PSD of $\mathbf{n}$ times the square of the magnitude of the CF's frequency response. This allows us to represent the $var(c_n)$ as

$$\begin{aligned} var(c_n) &= \sum_{i=0}^{d-1} S_c[i] \\ &= \sum_{i=0}^{d-1} \hat{\mathbf{p}}[i]|\hat{\mathbf{h}}[i]|^2 \\ &= \hat{\mathbf{h}}^T\hat{\mathbf{P}}\hat{\mathbf{h}} \end{aligned} \tag{2.10}$$

where d is the dimension of $\hat{\mathbf{h}}$, $\hat{\mathbf{p}}$ represents the PSD of noise $\mathbf{n}$, and $\hat{\mathbf{P}}$ is a diagonal matrix containing the elements of $\hat{\mathbf{p}}$ along its diagonal. Using Parseval's theorem [8], we can transform the original peak constraint $\mathbf{h}^T\mathbf{x}_i = c_i$ into the frequency domain

$$\begin{aligned} c_i &= \mathbf{h}^T\mathbf{x}_i \\ &= \frac{1}{d}\hat{\mathbf{h}}^\dagger\hat{\mathbf{x}}_i \\ &= \frac{1}{d}u_i \end{aligned} \tag{2.11}$$

and therefore $u_i = dc_i$. Minimizing the ONV from Eq. 2.10 subject to the peak constraint $\hat{\mathbf{X}}^\dagger \hat{\mathbf{h}} = \mathbf{u}$ results in a quadratic optimization with linear constraints, i.e.,

$$\begin{aligned} \min_{\hat{\mathbf{h}}} \quad & \hat{\mathbf{h}}^T \hat{\mathbf{P}} \hat{\mathbf{h}} \\ s.t. \quad & \hat{\mathbf{X}}^\dagger \hat{\mathbf{h}} = \mathbf{u}. \end{aligned} \tag{2.12}$$

Eq. 2.12 can be solved using Lagrangian multipliers (discussed in Appendix A), which produces

$$\hat{\mathbf{h}} = \hat{\mathbf{P}}^{-1} \hat{\mathbf{X}} (\hat{\mathbf{X}}^\dagger \hat{\mathbf{P}}^{-1} \hat{\mathbf{X}})^{-1} \mathbf{u}. \tag{2.13}$$

As noted earlier, when the noise is white, $\hat{\mathbf{P}} = \alpha \mathbf{I}$ and (assuming $\alpha \neq 0$) the solution in Eq. 2.13 simplifies to the ECPSDF filter shown in Eq. 2.8.

While the MVSDF filter is explicitly designed to handle noise, it still is not designed to produce sharp peaks. In order to design a filter that produces sharp peaks, we must attempt to control the entire correlation output plane. Both the ECPSDF and MVSDF filters constrain only the correlation value at the origin, thus there is no guarantee it will be easily distinguishable from the sidelobes. The Minimum Average Correlation Energy (MACE) filter addresses this directly.

2.2.2 Minimum Average Correlation Energy Filter

The MACE filter was introduced in 1987 by Mahalanobis et al. [9]. The MACE filter, unlike previous CFs, aims to control the entire correlation plane output $\hat{\mathbf{g}}_i$, and not just the correlation peak value $\hat{\mathbf{h}}^\dagger \hat{\mathbf{x}}_i$. This is done by minimizing the average correlation energy (ACE). For a given correlation output plane $\mathbf{g}_i$, the correlation energy is defined as $E_i = \mathbf{g}_i^T \mathbf{g}_i = \frac{1}{d} \hat{\mathbf{g}}_i^\dagger \hat{\mathbf{g}}_i$ (per Parseval's theorem). The ACE across all correlation output planes $\mathbf{g}_1, \mathbf{g}_2, \ldots, \mathbf{g}_n$ is defined as

$$
\begin{aligned}
\text{ACE} &= \frac{1}{n}\sum_{i=1}^{n} E_i \\
&= \frac{1}{nd}\sum_{i=1}^{n} \hat{\mathbf{g}}_i^{\dagger}\hat{\mathbf{g}}_i \\
&= \frac{1}{nd}\sum_{i=1}^{n} \hat{\mathbf{h}}^{\dagger}\hat{\mathbf{X}}_i\hat{\mathbf{X}}_i^{*}\hat{\mathbf{h}} \\
&= \frac{1}{nd}\sum_{i=1}^{n} \hat{\mathbf{h}}^{\dagger}\hat{\mathbf{D}}_i\hat{\mathbf{h}} \\
&= \hat{\mathbf{h}}^{\dagger}\left(\frac{1}{nd}\sum_{i=1}^{n}\hat{\mathbf{D}}_i\right)\hat{\mathbf{h}} \\
&= \hat{\mathbf{h}}^{\dagger}\hat{\mathbf{D}}\hat{\mathbf{h}}
\end{aligned}
\tag{2.14}
$$

where $\hat{\mathbf{D}}_i = \hat{\mathbf{X}}_i\hat{\mathbf{X}}_i^{*}$ is a diagonal matrix containing the power spectrum of $\hat{\mathbf{x}}_i$ along its diagonal, and $\hat{\mathbf{D}} = \frac{1}{n}\sum_{i=1}^{n}\hat{\mathbf{D}}_i$ is a diagonal matrix with the average power spectral density of the training images along its diagonal, and $\frac{1}{d}$ is a scaling factor accounting for the fact that an inner product in the space domain is scaled in the frequency domain by a factor of $\frac{1}{d}$, where d is the dimension of $\mathbf{x}_i$.

Minimizing the ACE while constraining the peak will lead to the following optimization:

$$
\begin{aligned}
\min_{\hat{\mathbf{h}}} \quad & \hat{\mathbf{h}}^{T}\hat{\mathbf{D}}\hat{\mathbf{h}} \\
s.t. \quad & \hat{\mathbf{X}}^{\dagger}\hat{\mathbf{h}} = \mathbf{u}.
\end{aligned}
\tag{2.15}
$$

We can observe that this is the same optimization problem as the MVSDF filter problem in Eq. 2.12 and can be solved in the same manner using Lagrangian multipliers (discussed in Appendix A), which produces

$$
\hat{\mathbf{h}} = \hat{\mathbf{D}}^{-1}\hat{\mathbf{X}}(\hat{\mathbf{X}}^{\dagger}\hat{\mathbf{D}}^{-1}\hat{\mathbf{X}})^{-1}\mathbf{u}. \tag{2.16}
$$

In practice, the filter $\hat{\mathbf{h}}$ can be computed efficiently by taking advantage of $\hat{\mathbf{D}}$'s diagonal structure.

In contrast to the previous ECPSDF and MVSDF filters, the MACE filter is much more likely to produce sharp peaks. Minimizing the ACE effectively suppresses the sidelobes of the correlation peak to be as small as possible; an optimal correlation output would be a delta-like function at the location of the target. The peak sharpness, relative to previous filter designs, makes precise localization more likely. However, there are drawbacks to the MACE filter design. The MACE filter generally emphasizes the high frequency components of the training samples, but in many cases (such as natural images), the test samples will have less energy in the higher frequencies. This in turn makes the filter more sensitive to distortions and high frequency noise. These properties are contrary to those seen in the MVSDF filter, which is designed to be more tolerant to noise but does not produce sharp peaks. Ideally, a filter could leverage the positive properties of both the MVSDF and MACE filters in a balanced manner. The Optimal Tradeoff Synthetic Discriminant Function (OTSDF) filter [10, 11] aims to do that.

2.2.3 Optimal Tradeoff Synthetic Discriminant Function Filter

The OTSDF filter was first introduced by Réfrégier and Figue in 1990 [10, 11]. The OTSDF filter balances the criteria optimized in two previous CF designs: the ACE criterion minimized when building a MACE filter [9], and the ONV criterion previously addressed by the MVSDF filter. The OTSDF filter minimizes the ACE for a given ONV, or vice versa.

As shown previously in Eqs. 2.10 and 2.14, ONV and ACE are given by $E_{ONV} = \hat{\mathbf{h}}^T\hat{\mathbf{P}}\hat{\mathbf{h}}$ and $E_{ACE} = \hat{\mathbf{h}}^T\hat{\mathbf{D}}\hat{\mathbf{h}}$. It has been previously shown that the optimal tradeoff between these two criteria can be found simply by minimizing the weighted sum of E_{ONV} and E_{ACE} [12], i.e.,

$$\min_{\hat{\mathbf{h}}} \qquad \hat{\mathbf{h}}^T\hat{\mathbf{D}}\hat{\mathbf{h}} + \beta\hat{\mathbf{h}}^T\hat{\mathbf{P}}\hat{\mathbf{h}}$$

$$s.t. \qquad \hat{\mathbf{X}}^{\dagger}\hat{\mathbf{h}} = \mathbf{u} \tag{2.17}$$

where $0 \leq \beta \leq \infty$ represents the tradeoff parameter between E_{ONV} and E_{ACE}. When $\beta = 0$, the result is the MACE filter, and as $\beta \to \infty$, the result becomes the MVSDF filter. We can define $\hat{\mathbf{T}} = \hat{\mathbf{D}} + \beta\hat{\mathbf{P}}$ and simplify Eq. 2.17 to

$$\begin{aligned} \min_{\hat{\mathbf{h}}} \quad & \hat{\mathbf{h}}^T\hat{\mathbf{T}}\hat{\mathbf{h}} \\ s.t. \quad & \hat{\mathbf{X}}^{\dagger}\hat{\mathbf{h}} = \mathbf{u}, \end{aligned} \tag{2.18}$$

and we see this has the same form as Eqs. 2.12 and 2.15. Again, by solving with Lagrangian multipliers (see Appendix A), we can obtain the closed-form expression for the filter,

$$\hat{\mathbf{h}} = \hat{\mathbf{T}}^{-1}\hat{\mathbf{X}}(\hat{\mathbf{X}}^{\dagger}\hat{\mathbf{T}}^{-1}\hat{\mathbf{X}})^{-1}\mathbf{u}. \tag{2.19}$$

We can replace β with a bounded variable by setting $\beta = \frac{1}{\lambda}(1-\lambda)$, where $0 \leq \lambda \leq 1$, i.e.,

$$\begin{aligned} \hat{\mathbf{T}} &= \hat{\mathbf{D}} + \beta\hat{\mathbf{P}} \\ &= \hat{\mathbf{D}} + \frac{1}{\lambda}(1-\lambda)\hat{\mathbf{P}} \\ &= \frac{1}{\lambda}(\lambda\hat{\mathbf{D}} + (1-\lambda)\hat{\mathbf{P}}) \\ &\propto \lambda\hat{\mathbf{D}} + (1-\lambda)\hat{\mathbf{P}}, \end{aligned} \tag{2.20}$$

where the $\frac{1}{\lambda}$ term can be ignored, as it can be shown that it does not affect the optimal filter. When we perform this variable substitution, choosing $\lambda = 0$ results in the MVSDF filter and $\lambda = 1$ results in the MACE filter. In many cases, choosing λ close to 1 is desirable. Such OTSDF filters will emphasize producing a sharp peak while also introducing robustness to noise that is not present in the MACE. These properties make the OTSDF filter effective for localization and detection tasks compared to filters previously introduced.

2.2.4 Maximum Margin Correlation Filters

The Maxmimum Margin Correlation Filter (MMCF) was introduced by Rodriguez et al. in 2012 [13], and combines the max-margin or generalization capability of support vector machines (SVMs) and the localization ability of CFs.

The SVM approach involves maximizing the margin between a hyperplane and training data samples by solving

$$\begin{aligned} \min_{\mathbf{w},b} \quad & \mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{N}\xi_i \\ s.t. \quad & t_i(\mathbf{w}^T\mathbf{x}_i + b) \geq c_i - \xi_i \end{aligned} \tag{2.21}$$

where we assume that we have N training samples $\mathbf{x}_i$ from two classes, each of length $d \times 1$. The vector $\mathbf{w}$ represents the hyperplane that SVM produces, and b represents the bias term. C represents the soft-margin penalty parameter, and ξ_i represents the slack variables of the SVM to account for the fact that training data may not be linearly separable. The variables $t_i \in \{1, -1\}$ and $c_i \in \{1, 0\}$ are defined according to whether $\mathbf{x}_i$ is a positive-class or negative-class training example, respectively. We can also express the formulation in the frequency domain as

$$\begin{aligned} \min_{\hat{\mathbf{w}},b'} \quad & \hat{\mathbf{w}}^\dagger\hat{\mathbf{w}} + C\sum_{i=1}^{N}\xi_i \\ s.t. \quad & t_i(\hat{\mathbf{w}}^\dagger\hat{\mathbf{x}}_i + b') \geq c_i - \xi_i \end{aligned} \tag{2.22}$$

where $b' = b \cdot d$.

The CF/localization criterion minimizes the mean squared error (MSE) between a desired correlation plane and the correlation output of the learned filter $\mathbf{w}$ and a training

sample $\mathbf{x}_i$ according to

$$\mathbf{w} = \arg\min_{\mathbf{w}} \sum_{i=1}^{N} \|\mathbf{w} \otimes \mathbf{x}_i - \mathbf{g}_i\|_2^2 \tag{2.23}$$

where the $\otimes$ symbol represents the correlation operation, and $\mathbf{g}_i$ represents the desired correlation output of the ith training image. Note that we are representing images and their 2D Fourier transforms by column vectors, through a lexicographic mapping from 2D to 1D. Eq. 2.23 can be expressed in the frequency domain as

$$\begin{aligned} \hat{\mathbf{w}} &= \arg\min_{\hat{\mathbf{w}}} \frac{1}{d} \sum_{i=1}^{N} \|\hat{\mathbf{X}}_i^* \hat{\mathbf{w}} - \hat{\mathbf{g}}_i\|_2^2 \\ &\propto \arg\min_{\hat{\mathbf{w}}} \sum_{i=1}^{N} \|\hat{\mathbf{X}}_i^* \hat{\mathbf{w}} - \hat{\mathbf{g}}_i\|_2^2 \end{aligned} \tag{2.24}$$

where $\hat{\mathbf{X}}_i$ is a diagonal matrix whose entries are the elements of $\hat{\mathbf{x}}_i$. For the correlation filter $\mathbf{w}$ to provide good localization performance, we want the correlation output at the center of the target to be large and small elsewhere. Therefore, we define $\mathbf{g}_i = [0, \ldots, 0, \mathbf{w}^T \mathbf{x}_i, 0, \ldots, 0]^T$, where the value of $\mathbf{w}^T \mathbf{x}_i$ is the desired output when the object is centered. The choice of $\mathbf{w}^T \mathbf{x}_i$ is for convenience. Accordingly, the frequency representation $\hat{\mathbf{g}}_i$ can be expressed as

$$\begin{aligned} \hat{\mathbf{g}}_i &= \mathbf{1} \mathbf{x}_i^T \mathbf{w} \\ &= \frac{1}{d} \mathbf{1} \hat{\mathbf{x}}_i^\dagger \hat{\mathbf{w}} \end{aligned} \tag{2.25}$$

where $\mathbf{1}$ is a $d \times 1$ vector of ones. We can expand Eq. 2.24 to get the following:

$$\begin{aligned} \hat{\mathbf{w}} &= \arg\min_{\hat{\mathbf{w}}} \sum_{i=1}^{N} \|\hat{\mathbf{X}}_i^* \hat{\mathbf{w}} - \hat{\mathbf{g}}_i\|_2^2 \\ &= \arg\min_{\hat{\mathbf{w}}} \sum_{i=1}^{N} \left(\hat{\mathbf{w}}^\dagger \hat{\mathbf{X}}_i \hat{\mathbf{X}}_i^\dagger \hat{\mathbf{w}} - 2\hat{\mathbf{w}}^\dagger \hat{\mathbf{X}}_i \hat{\mathbf{g}}_i + \hat{\mathbf{g}}_i^\dagger \hat{\mathbf{g}}_i \right) \end{aligned}$$

$$
\begin{aligned}
&= \arg\min_{\hat{\mathbf{w}}} \sum_{i=1}^{N} \left(\hat{\mathbf{w}}^{\dagger} \hat{\mathbf{X}}_i \hat{\mathbf{X}}_i^{\dagger} \hat{\mathbf{w}} - \frac{2}{d} \hat{\mathbf{w}}^{\dagger} \hat{\mathbf{X}}_i \mathbf{1} \hat{\mathbf{x}}_i^{\dagger} \hat{\mathbf{w}} + \frac{1}{d^2} \hat{\mathbf{w}}^{\dagger} \hat{\mathbf{x}}_i \mathbf{1}^{\dagger} \mathbf{1} \hat{\mathbf{x}}_i^{\dagger} \hat{\mathbf{w}} \right) \\
&= \arg\min_{\hat{\mathbf{w}}} \sum_{i=1}^{N} \left(\hat{\mathbf{w}}^{\dagger} \hat{\mathbf{X}}_i \hat{\mathbf{X}}_i^{\dagger} \hat{\mathbf{w}} - \frac{2}{d} \hat{\mathbf{w}}^{\dagger} \hat{\mathbf{x}}_i \hat{\mathbf{x}}_i^{\dagger} \hat{\mathbf{w}} + \frac{1}{d} \hat{\mathbf{w}}^{\dagger} \hat{\mathbf{x}}_i \hat{\mathbf{x}}_i^{\dagger} \hat{\mathbf{w}} \right) \\
&= \arg\min_{\hat{\mathbf{w}}} \hat{\mathbf{w}}^{\dagger} \left(\sum_{i=1}^{N} \hat{\mathbf{X}}_i \hat{\mathbf{X}}_i^{\dagger} - \frac{1}{d} \sum_{i=1}^{N} \hat{\mathbf{x}}_i \hat{\mathbf{x}}_i^{\dagger} \right) \hat{\mathbf{w}} = \arg\min_{\hat{\mathbf{w}}} \hat{\mathbf{w}}^{\dagger} \hat{\mathbf{Z}} \hat{\mathbf{w}}
\end{aligned} \tag{2.26}
$$

where

$$
\begin{aligned}
\hat{\mathbf{Z}} &= \sum_{i=1}^{n} \hat{\mathbf{X}}_i \hat{\mathbf{X}}_i^{*} - \frac{1}{d} \sum_{i=1}^{n} \hat{\mathbf{x}}_i \hat{\mathbf{x}}_i^{\dagger} \\
&= \hat{\mathbf{D}} - \hat{\mathbf{Y}} \hat{\mathbf{Y}}^{\dagger}
\end{aligned} \tag{2.27}
$$

where $\hat{\mathbf{D}} = \sum_{i=1}^{n} \hat{\mathbf{X}}_i \hat{\mathbf{X}}_i^{*}$ is a diagonal matrix, and $\hat{\mathbf{Y}} = \frac{1}{\sqrt{d}} [\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_N]$.

The goal of MMCF is to combine the criteria from Eqs. 2.22 and 2.26. The MMCF multi-criteria optimization problem is as follows:

$$
\begin{aligned}
\min_{\hat{\mathbf{w}}, b'} \quad & \left(\hat{\mathbf{w}}^{\dagger} \hat{\mathbf{w}} + C \sum_{i=1}^{N} \xi_i, \hat{\mathbf{w}}^{\dagger} \hat{\mathbf{Z}} \hat{\mathbf{w}} \right) \\
s.t. \quad & t_i (\hat{\mathbf{w}}^{\dagger} \hat{\mathbf{x}}_i + b') \geq c_i - \xi_i
\end{aligned} \tag{2.28}
$$

which can be written as a weighted sum of the two terms as

$$
\begin{aligned}
\min_{\hat{\mathbf{w}}, b'} \quad & \left(\lambda \hat{\mathbf{w}}^{\dagger} \hat{\mathbf{w}} + C' \sum_{i=1}^{N} \xi_i + (1 - \lambda) \hat{\mathbf{w}}^{\dagger} \hat{\mathbf{Z}} \hat{\mathbf{w}} \right) \\
s.t. \quad & t_i (\hat{\mathbf{w}}^{\dagger} \hat{\mathbf{x}}_i + b') \geq c_i - \xi_i
\end{aligned} \tag{2.29}
$$

where $0 < \lambda \leq 1$ and $C' = \lambda C$. The two quadratic terms can be combined to simplify Eq. 2.29 as

$$\min_{\hat{\mathbf{w}},b'} \quad \left(\hat{\mathbf{w}}^{\dagger}\hat{\mathbf{S}}\hat{\mathbf{w}} + C'\sum_{i=1}^{N}\xi_i\right)$$
$$s.t. \quad t_i(\hat{\mathbf{w}}^{\dagger}\hat{\mathbf{x}}_i + b') \geq c_i - \xi_i \tag{2.30}$$

where $\hat{\mathbf{S}} = \lambda\mathbf{I} + (1-\lambda)\hat{\mathbf{Z}}$.

The solution to the MMCF problem can be obtained in the same manner as standard SVM by transforming the data. We note that the expression in Eq. 2.23, and thus $\hat{\mathbf{w}}^{\dagger}\hat{\mathbf{Z}}\hat{\mathbf{w}}$ (Eq. 2.26) are non-negative, and therefore $\hat{\mathbf{Z}}$ is positive semidefinite. It follows that $\hat{\mathbf{S}}$ is positive definite. Therefore, we can transform the data as $\bar{\mathbf{w}} = \hat{\mathbf{S}}^{\frac{1}{2}}\hat{\mathbf{w}}$ and $\bar{\mathbf{x}}_i = \hat{\mathbf{S}}^{-\frac{1}{2}}\hat{\mathbf{x}}_i$. We can rewrite the MMCF formulation as

$$\min_{\bar{\mathbf{w}},b'} \quad \left(\bar{\mathbf{w}}^{\dagger}\bar{\mathbf{w}} + C'\sum_{i=1}^{N}\xi_i\right)$$
$$s.t. \quad t_i(\bar{\mathbf{w}}^{\dagger}\bar{\mathbf{x}}_i + b') \geq c_i - \xi_i \tag{2.31}$$

which has the same form as the original SVM formulation.

2.3 Introduction to Correlation Filter Trackers

2.3.1 Minimum Output Sum of Squared Error Filter

The MOSSE filter was introduced by Bolme et al. [1] as a CF tailored for visual tracking tasks. As the name suggests, the MOSSE filter is designed to minimize the MSE of the actual correlation output and the desired correlation output.

The MOSSE filter can be expressed as follows:

$$\mathbf{w} = \arg\min_{\mathbf{w}} \frac{1}{N} \sum_{i=1}^{N} \|\mathbf{w} \otimes \mathbf{x}_i - \mathbf{g}_i\|_2^2 + \lambda \|\mathbf{w}\|_2^2 \tag{2.32}$$

where $\mathbf{w}$ is the CF, $\mathbf{x}_i$ is the $d \times 1$ vector version of the training example, and $\mathbf{g}_i$ is the $d \times 1$ desired correlation output for that training example. Typically, $\mathbf{g}_i$ is chosen to be a Gaussian function centered at the origin with a small σ for positive training examples, and all zeroes for negative training examples. The parameter λ is for regularization. Note that the original formulation in [1] does not include the regularization term; this is equivalent to $\lambda = 0$, but we include it for generalization of the following derivation.

We can express Eq. 2.32 in the frequency domain as

$$\begin{aligned}
\hat{\mathbf{w}} &= \arg\min_{\hat{\mathbf{w}}} \frac{1}{Nd} \sum_{i=1}^{N} \|\hat{\mathbf{X}}_i^* \hat{\mathbf{w}} - \hat{\mathbf{g}}_i\|_2^2 + \frac{\lambda}{d} \hat{\mathbf{w}}^\dagger \hat{\mathbf{w}} \\
&= \arg\min_{\hat{\mathbf{w}}} \frac{1}{Nd} \sum_{i=1}^{N} \left(\hat{\mathbf{w}}^\dagger \hat{\mathbf{X}}_i \hat{\mathbf{X}}_i^\dagger \hat{\mathbf{w}} - 2\hat{\mathbf{w}}^\dagger \hat{\mathbf{X}}_i \hat{\mathbf{g}}_i + \hat{\mathbf{g}}_i^\dagger \hat{\mathbf{g}}_i \right) + \frac{\lambda}{d} \hat{\mathbf{w}}^\dagger \hat{\mathbf{w}} \\
&= \arg\min_{\hat{\mathbf{w}}} \hat{\mathbf{w}}^\dagger \left(\frac{1}{Nd} \sum_{i=1}^{N} \hat{\mathbf{X}}_i \hat{\mathbf{X}}_i^\dagger \right) \hat{\mathbf{w}} - 2\hat{\mathbf{w}}^\dagger \left(\frac{1}{Nd} \sum_{i=1}^{N} \hat{\mathbf{X}}_i \hat{\mathbf{g}}_i \right) + \frac{1}{Nd} \sum_{i=1}^{N} \hat{\mathbf{g}}_i^\dagger \hat{\mathbf{g}}_i + \frac{\lambda}{d} \hat{\mathbf{w}}^\dagger \hat{\mathbf{w}}
\end{aligned} \tag{2.33}$$

and we can find the minimum by solving $\text{grad}(\hat{\mathbf{w}}) = \mathbf{0}$ as follows

$$\begin{aligned}
\text{grad}(\hat{\mathbf{w}}) = \mathbf{0} &\implies 2 \left(\frac{1}{Nd} \sum_{i=1}^{N} \hat{\mathbf{X}}_i \hat{\mathbf{X}}_i^\dagger \right) \hat{\mathbf{w}} - 2 \left(\frac{1}{Nd} \sum_{i=1}^{N} \hat{\mathbf{X}}_i \hat{\mathbf{g}}_i \right) + \frac{2\lambda}{d} \hat{\mathbf{w}} = \mathbf{0} \\
&\implies \left(\frac{1}{N} \sum_{i=1}^{N} \hat{\mathbf{X}}_i \hat{\mathbf{X}}_i^\dagger \right) \hat{\mathbf{w}} - \left(\frac{1}{N} \sum_{i=1}^{N} \hat{\mathbf{X}}_i \hat{\mathbf{g}}_i \right) + \lambda \hat{\mathbf{w}} = \mathbf{0} \\
&\implies \left(\frac{1}{N} \sum_{i=1}^{N} \hat{\mathbf{X}}_i \hat{\mathbf{X}}_i^\dagger \right) \hat{\mathbf{w}} + \lambda \hat{\mathbf{w}} = \left(\frac{1}{N} \sum_{i=1}^{N} \hat{\mathbf{X}}_i \hat{\mathbf{g}}_i \right) \\
&\implies \left(\frac{1}{N} \sum_{i=1}^{N} \hat{\mathbf{X}}_i \hat{\mathbf{X}}_i^\dagger + \lambda \mathbf{I} \right) \hat{\mathbf{w}} = \left(\frac{1}{N} \sum_{i=1}^{N} \hat{\mathbf{X}}_i \hat{\mathbf{g}}_i \right)
\end{aligned}$$

$$\hat{\mathbf{w}} = \left(\frac{1}{N} \sum_{i=1}^{N} \hat{\mathbf{X}}_i \hat{\mathbf{X}}_i^{\dagger} + \lambda \mathbf{I} \right)^{-1} \left(\frac{1}{N} \sum_{i=1}^{N} \hat{\mathbf{X}}_i \hat{\mathbf{g}}_i \right) \tag{2.34}$$

which we can see involves only element-wise operations (because $\hat{\mathbf{X}}_i$ is a diagonal matrix). The canonical form is written as

$$\hat{\mathbf{w}} = \frac{\sum_{i=1}^{N} \hat{\mathbf{x}}_i^* \odot \hat{\mathbf{g}}_i}{\sum_{i=1}^{N} \hat{\mathbf{x}}_i^* \odot \hat{\mathbf{x}}_i + \lambda} \tag{2.35}$$

where the $\odot$ operator represents the Hadamard product. Also note that the parameter λ has subsumed a factor of N.

When used in scenarios that require incremental learning, such as visual tracking tasks, the MOSSE filter can be updated as a linear combination of the previously learned filter and a filter built on the new training examples. Given a filter $\hat{\mathbf{w}}_N$ learned on the first N training examples, we can simply add an element to the summations required in Eq. 2.35. Assume that

$$\hat{\mathbf{w}}_N = \frac{\sum_{i=1}^{N} \hat{\mathbf{x}}_i^* \odot \hat{\mathbf{g}}_i}{\sum_{i=1}^{N} \hat{\mathbf{x}}_i^* \odot \hat{\mathbf{x}}_i + \lambda} = \frac{\hat{\mathbf{a}}_N}{\hat{\mathbf{b}}_N + \lambda}. \tag{2.36}$$

When a new input $\hat{\mathbf{x}}_{N+1}$ becomes available, we can update $\hat{\mathbf{a}}_N$ and $\hat{\mathbf{b}}_N$ with the following updates

$$\begin{aligned} \hat{\mathbf{a}}_{N+1} &= (1-\eta)\hat{\mathbf{a}}_N + \eta(\hat{\mathbf{x}}_{N+1}^* \odot \hat{\mathbf{g}}_{N+1}) \\ \hat{\mathbf{b}}_{N+1} &= (1-\eta)\hat{\mathbf{b}}_N + \eta(\hat{\mathbf{x}}_{N+1}^* \odot \hat{\mathbf{x}}_{N+1}) \end{aligned} \tag{2.37}$$

where $0 \leq \eta \leq 1$ is a parameter controlling the learning rate. Smaller values of η correspond to slow adaptation, whereas larger values of η correspond to more aggressive adaptation.

The MOSSE filter is used in visual tracking because the update scheme in Eq. 2.37 allows a tracking system to quickly update the target model. Bolme et al. provide a qualitative comparison of the MOSSE tracker's accuracy compared to other trackers at that

time, while boasting an impressive 669 FPS. At such fast speeds, the MOSSE filter became a viable starting point for subsequent trackers that could be developed to be more robust and accurate at the expense of speed while still remaining faster than real-time (30 FPS).

2.3.2 Kernelized Correlation Filter

Henriques et al. [2] build on the concept of the MOSSE filter [1] by extending the filter to non-linear correlation. Linear correlation between a CF template and a test image is the inner product of the template $\mathbf{w}$ with a test sample $\mathbf{z}$ for every possible shift of the test sample $\mathbf{z}$. Instead of computing the linear kernel function $\mathbf{w}^T\mathbf{z}$ at every shift of $\mathbf{z}$, KCF computes some non-linear kernel $\kappa(\mathbf{w}, \mathbf{z}) = \varphi^T(\mathbf{w})\varphi(\mathbf{z})$ where κ represents a kernel function that is equivalent to mapping $\mathbf{w}$ and $\mathbf{z}$ into a non-linear space with the lifting function $\varphi(\cdot)$.

In one sense, KCF can be viewed as a change away from linear correlation filters, but it can also be seen as an efficient way of solving and testing with kernel ridge regression when the training and testing data is structured in a particular way (i.e., a circulant matrix).

Henriques et al. derive KCF from the standard solution of kernelized ridge regression. For learning $\mathbf{w}$, we assume the training data $\mathbf{X} = [\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{d-1}]$ is a $d \times d$ matrix where $\mathbf{x}_k$ contains the same elements as $\mathbf{x}_0$ shifted by k elements. The solution to kernelized ridge regression is given by [14]:

$$\boldsymbol{\alpha} = (\mathbf{K} + \lambda\mathbf{I})^{-1}\mathbf{g} \tag{2.38}$$

where $\mathbf{K}$ is the kernel matrix such that $\mathbf{K}_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$, $\mathbf{I}$ is the identity matrix, λ is the regularization parameter, $\mathbf{g}$ is the desired correlation output, and $\boldsymbol{\alpha}$ are the dual-space coefficients. The dual-space coefficients allow us to rewrite the original template $\mathbf{w}$ as

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i \varphi(\mathbf{x}_i). \tag{2.39}$$

When the kernel function $\kappa(\mathbf{w}, \mathbf{x})$ treats all data elements equally, and kernel $\mathbf{K}$ and the coefficients $\boldsymbol{\alpha}$ can be computed efficiently in the Fourier domain as follows:

$$\hat{\boldsymbol{\alpha}}^* = \frac{\hat{\mathbf{g}}}{\hat{\mathbf{k}}^{\mathbf{xx}'} + \lambda} \tag{2.40}$$

where $\hat{\mathbf{k}}^{\mathbf{xx}'}$ represents the first row of the kernel matrix $\mathbf{K}$ which contains the kernel function computation of $\mathbf{x}_0$ with all possible shifts of another data sample denoted $\mathbf{x}'$; either $\mathbf{x}_0$ in the training phase, or some test sample $\mathbf{z}$ in the testing phase. This idea is getting closer to the use of the Fourier domain to efficiently compute linear correlation. With non-linear kernels, Henriques et al. show that all elements of $\hat{\mathbf{k}}^{\mathbf{xx}'}$ can be computed efficiently. As an example, the Gaussian kernel $\kappa(\mathbf{x}, \mathbf{x}') = \exp(-\frac{1}{\sigma^2}\|\mathbf{x} - \mathbf{x}'\|^2)$ can be computed as

$$\mathbf{k}^{\mathbf{xx}'} = \exp\left(-\frac{1}{\sigma^2}\left(\|\mathbf{x}\|^2 + \|\mathbf{x}'\|^2 - 2\mathcal{F}^{-1}(\hat{\mathbf{x}} \odot \hat{\mathbf{x}}'^*)\right)\right). \tag{2.41}$$

where $\mathcal{F}^{-1}$ represents the IDFT. Just as in the case of the linear kernel, computing the Gaussian kernel in the Fourier domain reduces the computational complexity, although there are additional DFT/IDFT operations called compared to the linear kernel. During training, this kernel is computed for learning the coefficients $\hat{\boldsymbol{\alpha}}^*$ as shown above in Eq. 2.40, and when testing, the correlation is computed as

$$\hat{\mathbf{g}}' = \hat{\boldsymbol{\alpha}} \odot \hat{\mathbf{k}}^{\mathbf{xz}} \tag{2.42}$$

where the IDFT of $\hat{\mathbf{g}}'$ will produce the non-linear correlation in the space domain.

The KCF tracker utilizes these non-linear kernels to achieve performance improvements over the MOSSE filter. It operates in a similar fashion, with an update scheme in the same spirit as Eq. 2.37. One important distinction between the MOSSE tracker and the KCF tracker comes from the fact that the MOSSE tracker derives and stores a correlation filter, while the KCF tracker computes and stores the dual space coefficients as well as, necessarily, the training examples. As a tracker continues through a video sequence, the tracker could retain multiple training images, but this would result in tracking becoming progressively slower as the computational demands in solving for the kernel matrix $\hat{\mathbf{K}}$, and subsequently $\hat{\boldsymbol{\alpha}}$ as shown in Eq. 2.40, grow with the number of images stored. Rather than attempt to store multiple distinct training images or to discard data from old frames entirely, the KCF

tracker stores a single training image $\tilde{\mathbf{x}}$ that is a linear combination of previous images, so that

$$\tilde{\mathbf{x}}_k = (1-\eta)\tilde{\mathbf{x}}_{k-1} + \eta \mathbf{x}_k. \tag{2.43}$$

where $\mathbf{x}_k$ is the image patch in the kth frame.

When the KCF tracker was first introduced, it exhibited better accuracy than other trackers at that time, while reporting speeds greater than 150 FPS [2]. This combination of accuracy and speed made it a popular tracker to improve upon in a number of ways . Finally, it is important to note that the KCF is largely a reformulation of the CSK tracker introduced earlier by Henriques et al. [3]. One of the biggest changes between the CSK and KCF trackers is the addition of multi-channel features, which were previously introduced for CFs [15, 16]. Henriques et al. incorporate multi-channel features as a straightforward way to improve performance, where each channel is treated independently, and the correlation planes of each channel at test time are summed.

2.3.3 A Simple Correlation Filter Tracker

Secs. 2.3.1 and 2.3.2 discussed the design principles of the two CFs that appear in almost all CFTs. In this section, we outline how either the MOSSE filter or KCF can be implemented within a simple tracker. The simple tracker explained in this section can be considered a baseline tracker that other CFTs modify, but many details are first seen in the KCF tracker. Modifications often include simply swapping out particular components of this generic tracker, but other trackers may modify larger portions of the tracking workflow; these changes are discussed at length in Sec. 2.4.

Recall from the problem statement in Sec. 1.1.1 that the input to a tracker, along with $\mathcal{I} = \{I_0, I_1, \ldots, I_n\}$, is only the first frame of a video with a rectangular bounding box $\mathbf{b}_0 = [x_0, y_0, w_0, h_0]$ denoting the target region, and the output of the tracker is rectangular bounding boxes $\mathbf{b}'_k = [x'_k, y'_k, w'_k, h'_k]$ denoting the target location estimates in the rest of

the frames of the video.

In the first frame, we extract features from the given bounding box. It is important to detail how this training is done. Typically in CF applications, the template and the test image are zero-padded when performing the FFT [5]. The zero-padding is assumed to remove the circular convolution effects introduced by the DFT [17]. However, most CFTs do not perform zero-padding before computing the FFT of any space domain templates or image patches. Instead, the CF is computed from a region larger than the actual target; in some trackers this padding results in a CF with a width and height $2\times$ larger than the original target scale $[w_0, h_0]$.[3] This extra padding is done in combination with applying windowing – usually a cosine window – to both reduce the impact of the circular convolution and to emphasize the features that are within the original target region (and within the target region, the windowing emphasizes the center of the target even more). This design decision has tradeoffs: the larger CF allows the tracker to implicitly learn against some background information, and the windowing does reduce aliasing affects, but they are not removed entirely (and zero-padding this larger CF would likely reduce the speed significantly). It's important to note that incorporating the background into the CF training makes more sense in the tracking application; unlike tasks like ATR or other single-image object detection tasks, we know we will have to distinguish the target from a similar background in subsequent frames; this is not the case in other detection tasks.

From the first frame, we have an initial CF to detect the target throughout the video. In subsequent frames, the tracking process can still remain relatively simple. An image patch centered on the previous estimated target location is extracted from the new frame. This patch is usually the same size as the padded CF. The same feature extraction and windowing is performed. We take the DFT of the feature representation of the target, and perform the correlation between the image patch and the CF. We take the IDFT of correlation output, and the center position of the target is determined by the maximum

[3]Some trackers will resize targets to a fixed size, regardless of original scale and/or aspect ratio.

correlation value in the correlation output plane. If the target's scale is estimated, it may be done at this stage, or it may be done jointly with the translation estimation of the target. More details regarding scale estimation can be seen in Sec. 2.4.2 (the KCF tracker does not perform any scale estimation). For trackers that perform redetection and/or failure detection, this is usually when that tracking element is exercised (the KCF tracker does not perform any redetection).

Following the estimated target position (and scale), the CF must be adapted to the new target information. At the new target location (and appropriate scale), a final image patch is extracted[4] and features are again extracted and windowed. This image information is incorporated into the filter design. The MOSSE and KCF filters are designed so that this can be done with a simple linear combination of the previous CF and a CF designed solely on the new detection, i.e., $\mathcal{T}_i = (1-\lambda)\mathcal{T}_{i-1} + \lambda\mathcal{T}_{new}$, where $\mathcal{T}$ denotes whatever filter design is used, and λ denotes the adaptation rate that balances the previously learned model $\mathcal{T}_i$ and the information from the new detection, denoted $\mathcal{T}_{new}$. The details of the update as well as the value of λ will vary from tracker to tracker as CF designs vary, but nearly all newer CFTs will update their filter model. The two-step process of detect-update will continue through the duration of the video. An overview of this entire system framework is shown in Fig. 2.1.

We note that there are certain design decisions that are precluded in the Online Tracking Benchmark (OTB) and Visual Object Tracking (VOT) benchmark challenges. Both benchmarks prohibit revising old detection outputs based on more recent frames. Additionally, specific pretrained models are not allowed on a per-video basis. However, there is no prohibition on tailoring certain tracker parameters (e.g., amount of padding, adaptation rate λ, or CF specific parameters) for the entire dataset.

[4]While there is likely a large amount of redundancy between the image patches extracted for the target location estimation and then the filter update, most trackers repeat the entire feature extraction process. This is a case that illustrates that most trackers submitted to the benchmark challenges are not optimized for speed.

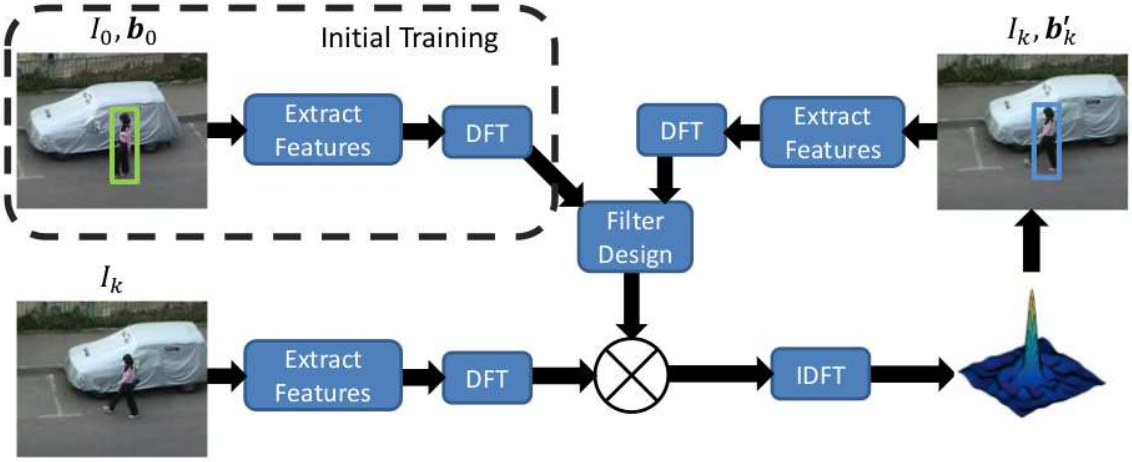


Figure 2.1: Flowchart of a basic CFT. I_0 and $\boldsymbol{b}_0$ refer to the first video frame and the ground truth bounding box for the target, I_k refers to a new frame in the video, and $\boldsymbol{b'}_k$ is the estimated bounding box of the target. The original MOSSE and KCF trackers follow this process, and can be considered a starting point for a more sophisticated tracking system.

2.4 Improvements for Correlation Filter Trackers

2.4.1 Feature Representations for Correlation Filter Trackers

Traditionally, CFs assumed scalar or single-channel features, e.g., grayscale intensities when operating on images. This was the case for much of the period prior to CFTs being introduced. In more recent years, CFs that accommodated other features, called vector CFs [15] or multi-channel CFs [16], have been developed. These CF designs were similar to the MOSSE formulation, in terms of minimizing the MSE of the correlation output plane. The work presented by both Boddeti et al. [15] and Galoogahi et al. [16] use the cross-spectral energy between different feature channels, and both choose histogram of oriented gradients (HOG) features to illustrate the new CF designs. As was discussed in Sec. 2.3.2, the KCF tracker uses multi-channel HOG features, but treats them independently. While treating each feature channel independently reduces the computation time, it ignores the possible interactions between feature channels and effectively assumes that all feature channels are independent. We note that the choice to treat each feature channel independently is the

prevailing choice in CFT designs.

Histograms of Oriented Gradients

From the original MOSSE and CSK trackers that used only scalar features, a number of feature descriptors have been explored in CFTs. As discussed above, the KCF tracker was the first to introduce HOG features to CFTs. HOG features were originally introduced for pedestrian detection but have become a popular feature descriptor in a range of object detection tasks [18]. HOG descriptors aim to capture edge features of a given target. Since the first use of HOG in CFTs, a number of subsequent CFTs use HOG, either as the only feature descriptor or in conjunction with other feature descriptors [2, 19–30].

The KCF tracker used a HOG cell size of 4×4 and retained all 31 feature channels, meaning that a HOG descriptor for a image patch of size $w \times h$ would be $\frac{w}{4} \times \frac{h}{4} \times 31$. Most trackers use the 4×4 cell size from the KCF tracker, but some trackers change this; there are CFTs that use a 1×1 cell size [19], a 6×6 cell size [20], and another that uses 2×2 cell size for small targets and the 4×4 cell size for larger targets [21]. Most trackers retain all 31 feature channels. The decision regarding cell size becomes a familiar tradeoff; smaller cell sizes produce denser feature descriptors, but reduce the speed of the tracker; using 31 HOG feature channels requires 31 FFTs. Larger cell sizes will keep the tracking speed much faster, but may not characterize the target well, particularly smaller targets. Overall though, HOG features can be computed quickly (independent of subsequent FFTs), perform much better than grayscale intensity features [2], and do not appear to slow tracking down at all when done at a cell size of 4×4 [2]. One final note regarding HOG features is that using a cell size of $c \times c$ means that the smallest detected target translations will be c pixels by default. The original KCF tracker does not address this, and therefore all estimated target translations are multiples of 4 pixels. A modified version of the KCF tracker uses sub-cell peak estimation to estimate target translations that are smaller than the HOG cell

size [31].

Color Features

While HOG descriptors capture the edge characteristics of a target, other features attempt to capture the color information of the target object. In many videos the target object has a distinctive color, e.g., a track and field athlete wearing a distinct jersey. If the target is the only yellow object in the video, it certainly seems straightforward to simply find the yellow blob in each frame. The Adaptive Color Tracker (ACT) was the first CFT to use color information for feature descriptors of the filter and target image patches [32]. The authors explore the effectiveness of a number of color spaces, e.g., RGB, LAB, HSV, and others. Their investigation shows that *color name* attributes [33] perform best. Color names are a higher dimensional representation of colors based on human perception. Unlike other color spaces which have a mathematical formula to convert from RGB color space, small ranges of RGB values are mapped to a probabilistic 11-dimensional vector that sums to 1, where each value corresponds to human perception of black, blue, brown, grey, green, orange, pink, purple, red, yellow, and white. Color names, like HOG descriptors, had previously been used in other computer vision tasks [34]. Since the publication of the ACT tracker, a number of CFTs have used color names jointly with HOG descriptors [23, 25, 28, 29]. The ACT tracker also proposes dimensionality reduction for the color names; this results in a 25% increase in the frames per second (FPS) while only slightly reducing accuracy.

While the color name features are a pixel-wise descriptor, other trackers use color information in a different manner. The Sum of Template And Pixel-wise LEarners (Staple) tracker [22] uses the first frame to learn which RGB values are representative of the target. In subsequent frames, per-pixel scores based on RGB values are smoothed out over a region equal to the target size to produce a color response plane. The amount of smoothing precludes a sharp peak from appearing within the color response, but it is used as a

complement to a CF built with HOG features. The HOG result will often produce a much sharper correlation peak, while the color information serves to reinforce or alter less sharp peaks that would correspond to less confident detections from the HOG features. The two output responses have different shape characteristics and are derived from different information (color vs. texture), and the overall result is stronger.

Much like HOG descriptors, color features are very quick to compute. Most color spaces have 3 channels, while the color names has 11 channels, which does result in slower performance (while HOG features often make up for 31 feature channels by reducing the spatial resolution, color features are typically of the same spatial resolution as the original target).

Deep-Learned Features

In recent years, deep convolutional neural networks (CNNs) have come to supplant "hand-crafted" features like HOG in computer vision tasks [35–37]. While hand-crafted features like HOG are computed based on what researchers expect to be salient feature outputs for discrimination, deep neural networks (NNs) are expected to learn discriminative features on their own, given sufficient training data. Just as deep features followed the introduction of a range of hand-crafted features in domains such as object classification and localization, CNNs are being introduced to visual tracking shortly after hand-crafted features.

The visual tracking problem is characterized by the lack of target data prior to the beginning of tracking. This immediately rules out training a CNN from scratch; instead, most CFTs that employ CNNs depend on a pretrained model, typically either AlexNet [35] or VGGNet [38].[5] At a high level, CNNs take an input image and, over successive layers of convolutions with filter banks and spatial pooling, learn feature representations that capture elements ranging from low-level textures to image classifications. For the simplest use in

[5]Although tracker parameters have often been optimized for a particular benchmark dataset, training an entire CNN model on the full dataset is explicitly disallowed from submissions to VOT.

visual tracking, the CNN can be considered a static feature extractor, similar to HOG or color features. This most basic approach is shown in work by Danelljan et al. [39], simply called the DeepCFT. The authors build a CF using the output from each convolutional network in VGGNet, which takes an input image patch of $224 \times 224 \times 3$ (all targets are resized to fit the pretrained network) and outputs a $109 \times 109 \times 96$ descriptor from the 1st convolutional layer, a $26 \times 26 \times 256$ descriptor from the 2nd convolutional layer, and $13 \times 13 \times 512$ descriptors from the 3rd, 4th, and 5th convolutional layers. The authors show that the best performance is obtained when building the CF using the 1st output layer, and in fact the performance drops off each successive layer until the 5th layer, which performs only 3rd best. The assumption is that the deeper layers do not provide enough spatial resolution; there is roughly a $17\times$ reduction in the spatial resolution from the original patch to the 3rd layer. Most importantly, the authors show that the CFs built from the CNN's output outperforms comparable CFs built jointly from HOG and color name features.

From this simple approach to building a deep CFT, more advancements have been made. Rather than just using the output from one layer, other works have combined the outputs from different layers [40–44]. The Multi-Level Deep Feature Tracker (MLDF) goes beyond just using a pretrained network and actually uses the current track information to train the CNN to adapt to the target appearance and its surroundings, rather than just keeping the starting VGGNet.

A look at recent benchmark performance shows that deep features are used in many of the most accurate trackers [45]. However, the use of deep features does come at the cost of speed; CFTs using CNNs are typically much slower than trackers using hand-crafted features. Despite this, visual tracking may follow a similar trend as other computer vision tasks that have become more and more dominated by deep networks.

2.4.2 Target Scale Estimation

In their simplest construction, CFTs simply estimate the translation of a target; the 2D correlation output plane provides no insight into the changing scale of a target. Accurately estimating target scale has multiple benefits; it directly affects how tracker performance is quantified in benchmark evaluations, and beyond benchmarks it can provide important information in real-world applications. Along with being valuable in and of itself, accurately estimating scale allows a tracker to adjust its own tracking procedure to adapt to the changing target, thus reducing the possibility of drifting off of or losing the target entirely. With the possible intrinsic and extrinsic benefits of accurate scale estimation, there has been a good deal of work in adapting CFTs to scale variation.

Exhaustive Scale Search

Perhaps the earliest CFT to address scale estimation was the Discriminative Scale Space Tracker (DSST), introduced by Danelljan et al. in 2014 [19]. DSST shares many similarities with the KCF tracker, but adds a scale estimation component following the translation estimation of the target. Following the translation estimation to determine $[x'_k, y'_k]$, the tracker extracts image patches at S scales. For each scale $n \in \left\{\lfloor -\frac{S-1}{2} \rfloor, \ldots, \lfloor \frac{S-1}{2} \rfloor\right\}$, DSST extracts an image patch of size $a^n w'_{k-1} \times a^n h'_{k-1}$, where a is the scaling factor between adjacent scales and $[w'_{k-1}, h'_{k-1}]$ is the previously estimated target size. Similar to the process for estimating the target translation, a separate CF designed for estimating scale is correlated with the feature descriptors extracted at each of the S scales (DSST chooses $S = 33$). Because the target centering is roughly accomplished, the correlation output is limited to a $S \times 1$ output.[6] Just as the CF for translation estimation is built in a way to favor smaller translations, a 1D Gaussian windowing is applied to the $S \times 1$ scale

[6]Producing a 1D correlation output for scale estimation in the image domain would be much less intensive than producing the prior 2D correlation output for translation estimation would be, but DSST still uses FFTs to produce the scale correlation output.

correlation output to favor smaller scale changes. This, along with a conservative scale factor of $a = 1.02$ result in small estimated scale changes from frame to frame, which is consistent with target behavior in most applications when the video's frame rate is 24-30 FPS (as is the case of nearly all benchmark videos). When the target scale is estimated, the scale filter is updated. In subsequent frames, the input image is resized according to the current scale for the estimation of the target translation. Finally, we note that while the translation filter in DSST uses 1×1 cell size HOG features, the scale filter uses PCA-HOG features [46] with 4×4 cells. The justification for using a larger cell size is that pixel-wise estimation is only a concern for the target translation.

DSST estimates the target bounding box in two steps, first by estimating the translation $[\Delta x, \Delta y]$ via a 2D correlation, and then jointly estimating $[\Delta w, \Delta h]$ by estimating the change in scale via a 1D correlation. The authors of DSST also explore estimating the translation and scale together by learning a 3D scale-space CF. They find that this 3D CF is much slower, as would be expected, and also actually performs slightly worse than the sequential translation and scale estimation. Following the findings during the development of DSST, a number of trackers have followed the approach of sequential translation and scale estimation [20, 22–26, 39]. Still, another CFT does jointly estimate translation and scale [47].

Efficient Scale Search

DSST and trackers that adopted the same approach look exhaustively over a range of possible scales, which may not be the most efficient approach possible. The Multi-Kernelized Correlation Filter (MKCF) tracker [28] seeks to estimate the scale by finding the scale that produces a corrrelation output peak with the highest PSR in a more efficient manner by performing a line search within a range of scales $\pm 10\%$ of the current scale. The assumption this approach makes is that the PSRs across different scales within the search range will

only have a single local maximum; with multiple local maxima, a suboptimal scale could be chosen. The authors of [28] do not report how often this assumption is accurate. In addition to a line search, MKCF chooses to rescale the features rather than extract features multiple times from different scales. The authors do compare this approach to the "traditional" approach of extracting features from image patches of different scale, and show a small gain in the accuracy when doing the faster rescaling of features, though this effect is quite small.

The above mentioned trackers estimate the scale either exhaustively or with a more efficient line search. The Multi-view Correlation Filter Tracker (MvCFT) [29] reduces the scale estimation to a discrete decision to decide if the target is getting smaller, remaining the same size, or getting larger. Once the target translation is estimated, image patches at the current scale and $\pm 5\%$ are tested against the same CF used in translation estimation. The maximum value from the correlation output plane for each of the 3 scales is taken, and, after the unchanged scale is given a small amount of extra weight, the maximum value of all 3 planes is used to determine if the target is getting smaller, larger, or remaining the same size. If the scale is changing, it is changed by 5%. It can be assumed that very small scale changes will not be detected, but CFTs without any scale estimation are robust to small changes, so effectively ignoring the smallest changes should not be a large concern, and the authors claim experimental results support this.

2.4.3 Parts-Based Correlation Filter Trackers

Parts-based models seek to perform vision tasks often by decomposing a large object into smaller pieces that can be operated on independently and then joining the results of the subproblems to provide a coherent result for the entire object. Parts-based models have been used previously in object alignment [48] and object detection [49]. More closely related to CFTs, recent work on object alignment used CFs to detect individual car parts before fusing the result with a deformation model [15]. Parts-based approaches are

designed to handle object deformations, e.g., a pedestrian's changing stride, whereas a single rectangular detection window may struggle to characterize all such possible poses of the target. Additionally, parts-models are inherently tolerant to partial occlusion; if some object parts are occluded, a detector can still work well based on the strength of the visible parts.

Visual trackers stand to benefit from some of the intrinsic characteristics of parts-models, but their application and benefits are not the same as those in single-image object detection. Parts-based object detectors can benefit from target knowledge and possible deformations, e.g., a pedestrian detector can be designed to have a part for each limb. With no prior knowledge of the target, parts cannot be defined so precisely for trackers; instead a generic parts configuration is necessary. Still, visual trackers will benefit from robustness to occlusion (for more discussion regarding occlusion handling in visual trackers, see Ch. 5). Additionally, parts-models can easily be tailored to estimate scale; if parts are drifting farther apart, that alone can be enough to indicate that the scale is increasing.

A tracker proposed by Liu et al. [27] uses KCF filters to localize 5 object parts, each approximately 20% the total target height and width, configured in a cross pattern (see Fig. 2.2). The individual part detections are given weights according to two factors: a higher PSR for a part's detector will result in higher weight, and a smaller shift from the previous location will result in a higher weight. The use of PSR is mostly self-explanatory, but emphasizing smaller shifts does require justification. The authors argue that detectors for parts that become occluded can possibly detect another unoccluded part of the object; this is part of the risk of defining a generic parts-model for all possible targets. More generally, the justification is that if all part detectors shift equally between frames, it is likely the entire object did, and there will be no net effect of this shifting penalty on the relative weights between parts. If 1 of the 5 part detectors shifts much more than the other parts, it is more likely an error and should be given less weight (although, under this design, an

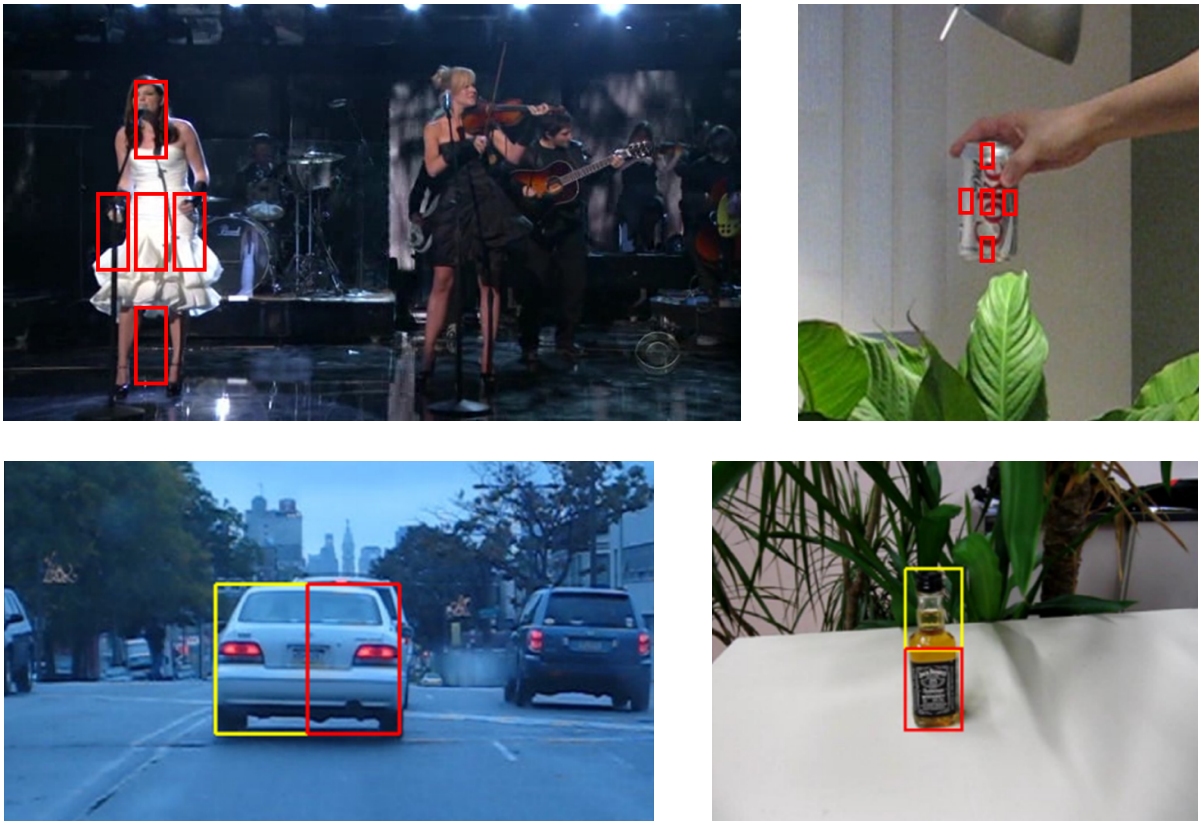


Figure 2.2: Configuration of parts used in [27] (top) and [21] (bottom). Liu et al. use 5 smaller parts that do not collectively cover the entire target, while Akin et al. use 2 parts, configured according to the target's aspect ratio, that cover the entire target. Figures are adapted from [27] (top) and [21] (bottom).

outlier part could possibly shift *less*; this is not addressed). Once the individual correlation planes are weighted, they are combined to provide a full confidence map, and the final target translation and scale estimation is determined by Bayesian inference similar to a previous tracker [50].

The tracker proposed by Akin et al. [21] subdivides targets into only 2 parts: either top and bottom parts for tall and narrow targets or left and right parts for short and wide targets (see Fig. 2.2). The two parts use KCFs filters to localize their half of the target.The reliability of the 2 part detectors is indicated by the correlation peak value for each part. Based on these weights, an additional full-target KCF is built as well and performs a full-target target detection centered at the location suggested by the two part detectors.

The target's scale is estimated by measuring the changing distance between the two parts. When updating the CF models, the correlation peaks are tested against a threshold; if a part detection falls below some threshold, the part model will not update, and if all parts fall below the threshold, the full-target detector will not update. This is meant to avoid updating the models when the target or target part is occluded. Additionally, scale estimation is only performed when all part detections are considered reliable.

2.5 Other Visual Trackers

In Sec. 2.4, we discussed a wide range of CFTs and the various improvements they make to the MOSSE and KCF trackers that first used CFs for visual tracking. However, both prior to the introduction of the MOSSE tracker and during the continued growth of CFTs in recent years, many other visual trackers that do not use CFs have been developed.

One of the most well-known trackers is the Tracking-Learning-Detection (TLD) tracker introduced in 2012 [51]. As its name suggests, the tracker has three components. A Median-Flow tracker [52] locates the target from frame to frame based on the current trajectory. The detector treats new frames independently of previous frames and can correct failed track. The learner observes both the tracker and detector, and estimates when the detector is making errors. Based on when the learner believes the detector is making errors, it can generate more training data for the detector to improve its performance. The TLD tracker was the third most accurate tracker when the OTB50 benchmark was published in 2013 with 29 trackers included [53]. The TLD tracker is not evaluated on the most recent VOT benchmarks, but a more recent proposed tracker that fuses the principles of both the TLD and KCF trackers has been proposed [54].

The best performing tracker in the OTB50 benchmark's collection of trackers was the Sparsity-based Collaborative Model (SCM) tracker [55]. Zhong et al. introduce a discriminative and a generative model which learn sparse grayscale features and sparsity-

based histograms, respectively, within a particle filter framework. The SCM combined the two approaches while stating that the generative model plays a more significant role in the tracking. The second best tracker reported in the original OTB50 benchmark was *Struck* [56]. Struck trains a structured output kernel SVM that continuously adds previous detections as well as hard negatives from the region around each detection, while also pruning the number of possible support vectors over time to avoid progressively slower processing times.

Despite the success of the above trackers, a large number of new trackers have been introduced since the OTB and VOT benchmarks essentially regulated the ways trackers operate and are evaluated; nearly all of the most effective trackers on these benchmarks have been developed since the introduction of these benchmarks (and perhaps developed explicitly for the challenges present in the datasets). This is true for CFTs and other trackers.

The best performing tracker on the VOT2015 dataset was the Multi-Domain Network (MDNet) tracker [57]. MDNet pretrained a CNN on an outside set of videos, then combines this pretrained network with a binary classification layer for a test video. Candidate regions are sampled with varying translations and scale changes relative to the previous target detection. The significance of MDNet is that its success, coupled with being one of the first trackers to use CNNs, likely inspired a growing number of newer trackers that use deep networks. Other CNN trackers include an extended version of the MDNet tracker, with occlusion inference and a scale regression model to refine the output bounding boxes, submitted to the VOT2016 benchmark [45], and the Tree-Structured Convolutional Neural Network (TCNN) tracker [58] that uses a CNN along with a tree structure to capture the multi-modal appearance of certain targets. Another recent tracker exchanges the CNN for a Siamese NN [59].

While deep networks are growing in popularity, there are other trackers which also

performed very well in recent benchmarks. The Edge Box Tracker (EBT) [60] uses an objectness measure to find region proposals within an entire frame, which can then be processed by any object detector. The tracker focuses on finding hard false-positives and re-ranking proposal regions, which can be processed separately. The Salient Region Based Tracker (SRBT) uses color information to segment a target more precisely than a rectangular bounding box; this more precise segmentation determines which regions of the rectangular bounding box contribute to the model update [45]. The Geometric Hypergraph Tracker (GGT) [61] uses a graph structure to capture the relationships between different parts of the target as correspondences between frames are found and used to find a subset of reliable parts. An extended version of GGT appears in VOT2016 that incorporates the Scale Adaptive with Multiple Features (SAMF) CFT.

Along with original tracking systems, the outputs from multiple trackers can be combined to produce one composite output. The Median Absolute Deviations (MAD) fusion strategy [62] is able to detect outliers, or trackers which have likely failed. The amount that each individual tracker output deviates from the median determines the weight given to that tracker for the final estimate, and outliers are ignored and also reinitialized on the new estimated target location. In the VOT2016 benchmark contest, it uses a swarm of KCF trackers and a DSST scale estimation scheme, and outperforms both KCF and DSST [45].

2.6 Chapter Summary

This chapter introduced fundamental CFs that are both still used and preceded the first CFs designed for visual tracking applications. We then introduced fundamental CFTs, and discussed a wide range of improvements to trackers that still use CFs at their core. Along with an extensive discussion of CFTs, we briefly discussed other non-CF trackers.

While the discussion of CFs is sufficient for giving context to CFTs as well as the subsequent chapters in this thesis, other sources are recommended for a more in-depth

treatment of CF theory and a wider range of CF designs [5, 63]. Additionally, visual tracking is a very active research field; we recommend referencing the latest versions of the VOT Challenge [64], which continues to be run on an annual cycle at the time of this publication. The VOT Challenge also includes a number of unpublished trackers which often include small modifications of previous trackers or borrows ideas from multiple existing trackers. Additionally, we reserve the discussion of some particular trackers that most closely relate to different CF design choices for trackers and handling occlusion for Chapters 4 and 5, respectively.

Chapter 3

Tracking Benchmarks and Evaluation

3.1 Testing Protocols

The visual trackers discussed in this work can be applied and evaluated on a vast amount of data. To avoid bias in evaluating the quality of different trackers, widely adopted datasets and statistical evaluations are reported throughout this work. In addition, we also propose a new approach to evaluating tracker outputs in an effort to measure the statistical significance of different tracker outputs.

3.1.1 Tracking Datasets

The trackers evaluated in this work are tested on two series of widely adopted benchmark datasets: the OTB and the VOT benchmark [45, 65]. Both datasets have had multiple iterations published, and their statistics are shown in Table 3.1. The original OTB dataset, denoted OTB50, contains 51 targets across 50 videos [53], while the extended version, denoted OTB100, contains 100 targets across 98 videos [65]. Three annual iterations of VOT, denoted VOT2013, VOT2014, and VOT2015, contain 16, 25, and 60 targets,

Dataset	OTB50	OTB100	VOT2013	VOT2014	VOT2015
Tracks	51	100	16	25	60
Segments	2175	4181	643	957	1722
Frames	29486	59016	6094	10209	20931

Table 3.1: Number of full-color videos, segments, and frames in the benchmark datasets. See Sec. 3.2 for details regarding the definition of segments.

respectively, in separate videos [31, 66, 67].[1]

Both series of benchmark datasets have a diverse collection of videos in terms of the targets, the environments, and the challenges in tracking the objects. There are a number of cars, persons, and faces in the datasets, along with other assorted objects and animals. The videos include scenes that could be captured by surveillance cameras, television broadcasts, and videos taken in lab settings to explicitly pose challenges to trackers following the object of interest. The OTB datasets include 11 tags[2] if a video has particular challenging attributes. These particular tags will be referenced when appropriate during analysis. The VOT benchmarks do not have explicit tags, but the VOT datasets contain similar challenges.

3.1.2 Tracking Benchmark Performance Measures

There are a number of statistical approaches to quantifying the performance of a tracker following a given target [68]. In general, there are two ways to qualitatively evaluate a tracker: the tracker can *drift* off the center of the target but still generally follow the movement of the target, or the tracker can *fail* or lose the location of the target entirely, at which point any overlap of the tracker and the target could be considered a chance event and the movement of the target is not expected to be reflected at all by corresponding

[1] A fourth iteration of the VOT benchmark in 2016 contained the same videos as the previous iteration with new ground truth annotations that included rotated bounding boxes. Because most of the trackers evaluated in this work do not produce rotated bounding boxes by design, we evaluated against the VOT2015 ground truth annotations.

[2] These tags are: illumination variation, scale variation, occlusion, deformation, motion blur, fast motion, in-plane rotation, out-of-plane rotation, out-of-view, background clutters, and low resolution.

movement of the tracker. For nearly any application, we would expect a *failed* tracker to be a more severe issue than a tracker that has only *drifted* partially off the target, but drifting can severely reduce the efficacy of post-tracking processing (e.g., face recognition on a tracked face) as well as likely making the tracker more vulnerable to failing later in the video (or beyond the stopping point of the video as included in the dataset).

In this work we will report results consistent with the OTB evaluations. The OTB reports statistics based on two measures: central location error (CLE) and region overlap (often referred to simply as overlap). CLE is simply the Euclidean distance between the marked (i.e., ground truth) center point and the estimated center point:

$$\mathrm{CLE}(\mathbf{b}_k, \mathbf{b}'_k) = \sqrt{(x_k - x'_k)^2 + (y_k - y'_k)^2} \tag{3.1}$$

Note that the CLE does not account for any estimated size of the target, and additionally that it is not normalized for the scale of the target. Overlap is defined as the intersection of the ground truth region and the estimated target region:

$$\mathrm{overlap}(\mathbf{b}_k, \mathbf{b}'_k) = \frac{|R_k \cap R'_k|}{|R_k \cup R'_k|} \tag{3.2}$$

where R_k and R'_k represent the set of pixels enclosed within the rectangular region defined by $\mathbf{b}_k$ and $\mathbf{b}'_k$, respectively, and $|\cdot|$ represents the number of pixels in the region. Unlike CLE, overlap does account for both the center location estimate as well as the estimated scale of the target, as shown in the examples in Fig. 3.1.

Another important distinction between CLE and overlap is that overlap has a useful lower bound of 0, representing zero overlap and a complete failure to locate the target, while the upper bound CLE is roughly determined by the dimensions of the frames for a given video. It is easily possible that two trackers, both with detections with no overlap with the true target and both completely failed, could have very different CLEs. Because of this, it is not informative to average the CLEs of all frames in a video to provide a summary performance statistic of a tracker, whereas averaging the overlap of a tracker with the ground truth can provide a valid comparison of two trackers' relative performance.

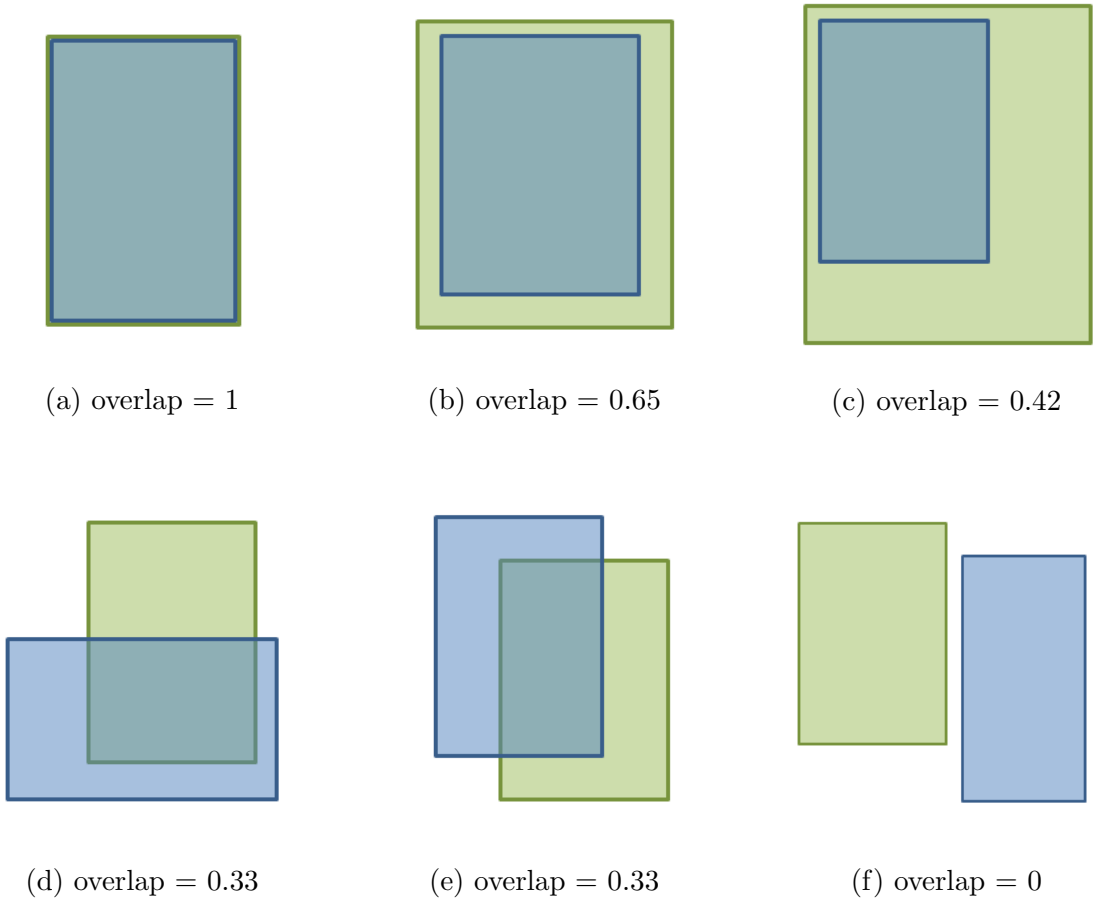


Figure 3.1: Example overlap calculations when the region estimate includes different scale and translation errors. These examples do not distinguish which box represents ground truth because this does not affect the calculation.

The OTB calculates statistics from the CLE and overlap measures. From the CLEs for a tracker on a single video, a precision plot reports for what percentage of frames the CLE is less than thresholds ranging from 0 to 50 pixels. As noted above, statistics for higher CLEs can be unreliable. In addition to precision plots, a single threshold value of 20 pixel error is often used to compare trackers. For overlap, a similar success plot is presented showing how often the tracker has overlap greater than thresholds spanning from 0% to 100%. A single value, the average overlap across all frames, is also reported. When these statistics are reported for results across multiple videos, all videos are given equal weight, regardless of differences in the video length.

Both CLE and overlap capture and reflect errors of drift and failure; they are sensitive to smaller changes and will be sharply affected by complete failures (and in the case of CLE, the effects will be unpredictable). The VOT benchmark evaluation scheme is different in a critical aspect: the trackers are run in a supervised manner. In this context, the supervision refers to the tracker being reinitialized when the tracker produces an estimate that has zero overlap with the ground truth. When this happens, the tracker is reinitialized 5 frames later in the video, and additionally the accuracy of the subsequent 10 frames after reinitialization are not used in evaluating the tracker. VOT uses two measures to quantify performance. The first is average overlap (referred to as "accuracy" in VOT), which is computed in the same way as in OTB, although because of reinitialization, this statistic more closely reflects the drift errors of the tracker. The second measure used by VOT is simply the number of times the tracker needs to be reinitialized, and corresponds to tracking failures (referred to as "robustness" in VOT).

We feel that both the OTB and VOT benchmarks, along with most works proposing new trackers, do not sufficiently address the issue of the statistical significance of performance measures between different trackers. While the VOT benchmark attempts to "test for practical difference" between trackers by measuring the variance of human annotators for different tracks [31], the reinitialization of trackers, even when discarding the first reinitialized outputs, makes any such measure difficult to interpret. More generally, measuring the statistical significance of tracking results is a challenge due to the significant dependence of consecutive frames and the difficulty of computing a fair measure of tracking accuracy at all. With those caveats, we do report the CLE and overlap statistics used by VOT (we do not run trackers in a supervised manner), but we do introduce a new measure that we feel is both intuitive and can allow for the ability to measure statistical significance of different tracking results.

3.2 A New Evaluation Metric: Average Segment Overlap

As noted above, the difficulty of quantifying tracking performance comes both from the different ways to measure accuracy, but also from the fact that each tracker output is not an independent decision, but rather a decision that is highly dependent on the decisions made on all previous frames. The OTB and VOT benchmarks try to capture both drift and failure errors, but in all cases, trying to determine statistical significance between results is challenging.

We propose a new measure, average segment overlap (ASO), for quantifying performance, as well as a statistical test to measure its statistical significance. The ASO depends on defining *video segments* (or simply "segments"), which can be considered as a part of the video where a single event (or multiple simultaneous events) happens to the target object, e.g., object(s) occlude the target, the illumination of the target changes, or the target significantly deforms and changes appearance. What constitutes a segment could be left to human annotators, but we discuss in Sec. 3.3 a proposed method to automatically segment a video based on the object we are tracking. The justification for using overlap per video segment, rather than overlap per frame, is that we feel that the length of a video is not as important as trying to quantify how many different events occur within the video. At an extreme case, a video can be decoded in such a way that two consecutive frames are identical; intuitively, these two frames are not collectively twice as important as any other given frame, and should not be given twice as much weight when evaluating trackers. A less extreme example, but one that occurs in the benchmarks datasets more frequently, is a case where the entire scene is relatively static, and the only difference between consecutive frames is due to video compression and decoding. This can easily happen when videos are created by researchers, as it can be perceived that the person that perturbs the tracked

object and its surroundings first turns on the video recorder, then takes some time to actually start moving objects and producing a challenging tracking problem. Whether this period lasts for 10 frames or 100 frames does not materially change the video, but the length of the segment will change the quantitative performance measures, which is not an appropriate reflection of the tracker's performance. In the instances where this occurs at the beginning of the video, there is at least an expectation that all trackers will be performing equally well; but for long stretches of no activity later in the video where some trackers have already failed, the difference in tracking performance is amplified by longer stretches of minimal activity.

To calculate the ASO for a given tracker $\mathcal{T}$ on video $\mathcal{I}$, let us assume that the video has p video segments $S_1, S_2, \ldots, S_p$. The ASO is defined as

$$ASO(\mathcal{I}, \mathcal{T}) = \frac{\sum_{i=1}^{p} \frac{\sum_{j \in S_i} \text{overlap}(\mathbf{b}_j, \mathbf{b}'_j)}{|S_i|}}{p} \tag{3.3}$$

where $|S_i|$ represents the number of frames within S_i. Note that ASO is identical to average overlap as defined by the OTB and VOT benchmarks when the video is considered to have a single segment.

When computing the ASO across multiple videos, we make the decision to *not* give equal weight to different videos with different numbers of segments. While giving every video equal weight is sensible when using average frame overlap, because of the tendency for longer videos to be less challenging per frame (some videos may actually be longer to include enough challenges for the tracker), segmenting the videos and using ASO is a direct attempt to quantify how many events happen within a video. When considering this underlying idea, it seems more reasonable to give more weight to videos that have more events happening within them.

To further motivate ASO as an alternative to average frame overlap, as used in tracking benchmarks, consider a contrived 100 frame video of a dark, stationary object against a textureless white background. During the video, only one thing happens: a distinct object

appears from off-camera, passes in front of the target object and briefly occludes it, and continues to move across the field of view until it passes off-camera opposite from where it entered the scene. After the object leaves the scene, the video continues for some period of time. We will assume that two trackers, denoted Tracker A and Tracker B, achieve perfect overlap prior to the occlusion occurring, and that once the occlusion occurs, Tracker A continues to track the target perfectly for the remainder of the video, while Tracker B actually begins to follow the occluding object, and shortly after the occlusion has zero overlap with the target object for the remainder of the video (for simplicity, we will assume that the performance degrades from perfect overlap to zero overlap in a single frame). When qualitatively evaluating the two trackers, an observer is likely more interested in discovering "how well did the trackers perform when the new object occluded the target object?" The evaluation is intuitively looking to see how a tracker handled an event, not how well the tracker performed on frame 1, 2, 3, . . . , 100. In this toy example, the trackers are really just being given a single "test," which trackers will either pass or fail (or pass with some additional drift being introduced). What is *not* particularly relevant is how long the video is before or after this "test." In the case of the successful tracker, the ASO would equal 1, because it was accurate before and after the occlusion. The failed tracker would have a ASO of 0.5, because it was tracking the target before the occlusion but not after.[3] In contrast, the average frame overlap for the failed tracker will be significantly impacted by the *timing* of the occlusion. The successful tracker will still have average frame overlap of 1 regardless of the timing of the occlusion, but the failed tracker could have a average frame overlap of 0.1, 0.5, or 0.9, depending if the occlusion occurs at frame 10, 50, or 90, respectively. In the case where the occlusion occurs very late in the video, this number is misleading and could make the performance between the trackers hard to distinguish. For

[3]There is a case to be made that results in the first video segment should be discarded. In the case of the example, it does not make sense that the failed tracker be rewarded for "tracking" a stationary, static object. However, in actual videos there are usually many more segments, thus reducing the impact of the initial segment, and additionally there is no guarantee a tracker will not fail before a significant event happens within the scene.

the issue illustrated in this example, we believe ASO to be more reflective of a tracker's performance.

In summary, we feel the balancing of accounting for both drift and failed track errors, as well as emphasizing a measure that reflects the tracker's ability to handle challenging events, via the pooling of frames into video segments, makes the ASO measure and associated test a fair and robust evaluation method for trackers.

While we introduce the ASO measure, we will still report results using the previously discussed CLE and average overlap measures. Because of the equal weighting given to each video (i.e., every video is considered a single segment) in the previously introduced benchmarks and to reduce ambiguity, we will henceforth refer to the existing overlap measure as average video overlap (AVO). In general, when there is a very large gap between the accuracy of different trackers, the rank-ordering of the trackers will be the same when computing any commonly used tracking accuracy metric. However, just as the rank-ordering of trackers may differ when CLE or AVO is used, the rank-ordering of trackers will change between that of the ASO and AVO (and CLE) metrics. Within this work, we can see examples of this and provide further discussion in Sec. 5.4.2.

3.2.1 Statistical Testing for Average Segment Overlap

One of the problems with other measures is that there was no suggested method for determining if new tracker results were better; it is instead just generally accepted that higher accuracy is better, with no attention given to the statistical significance of the results. While the ASO over an entire dataset is an informative measure, the aggregate statistic is not the optimal way to compare different trackers. In choosing a statistical test for two trackers, we should aim to maintain the dependence between the segment overlaps of different trackers, and so we will use paired t-tests. Using a paired test will favor a tracker that fixes the mistakes of the tracker it is being compared to without introducing new

errors; the relatvie accuracy of a test tracker that beats another baseline tracker on 10% of segments would have a lower variance than another test tracker that improves accuracy on 30% of segments while performing worse on a different 20% of segments compared to the same baseline tracker. This aligns with how a human choosing between two trackers might evaluate the choices: one tracker being an improved version of a baseline, with very little concern that certain use cases might become less reliable, and another tracker that may perform much better or may perform worse, depending on how well it is adapted to the particular application of interest.

3.3 Segmenting Videos

If ASO is the measure chosen, then a way of segmenting the videos must be decided. As stated above, human annotators could mark the beginning/end of segments in the video, but this process is difficult to reproduce with different annotators (and possibly with the same annotator over repeated annotations), and additionally, human annotators may not accurately perceive what will challenge tracking algorithms. None of the tracking videos in the OTB and VOT benchmarks are challenging for adults; it is perhaps unreasonable for a human to then suggest what a computer system will find challenging. Instead, we try to statistically determine when a video has changed sufficiently to warrant being a new segment (admittedly, the threshold corresponding to a "sufficient change" is chosen manually).

Based on our knowledge of trackers, we aim to define segments when the target object has changed in appearance and/or moved significantly. We begin by holding the first frame in the video as our *reference frame*, denoted I_{ref}. Additionally, we keep track of the defined ground truth region, denoted R_{ref}, for that frame. For each subsequent frame I_{new} with ground truth region R_{new}, we mask out the regions of I_{ref} and I_{new} that do not correspond to either R_{ref} or R_{new} and then compute the normalized cross-correlation between the

images:

$$\left\langle \frac{I_{ref}(R_{ref} \cup R_{new})}{\|I_{ref}\|}, \frac{I_{new}(R_{ref} \cup R_{new})}{\|I_{new}\|} \right\rangle \gtrless \xi \tag{3.4}$$

where $\langle \cdot, \cdot \rangle$ represents the correlation operation, and ξ is a decision threshold to determine if the ground truth regions are sufficiently different to end the current video segment and begin a new one. The regions used in this correlation operation are illustrated in Fig. 3.2. When performing the cross-correlation, we simply use gray-scale pixel intensities rather than the RGB values. This allows for consistent calculations between RGB videos and videos that are already gray-scale. Finally, when we do reach a new frame that is sufficiently different, we actually include the next 5 frames in the current video segment. The reason for this is that in many cases, the challenging factor that triggers the end of the video segment quickly leaves the ground truth region (e.g., an occluding object briefly passing in front of the target object). Without the 5 frame buffer, the video can be split into 3 parts: before, during, and after the occlusion. However, the occlusion is understood to be a single event, and thus it makes more sense that the video be only split into segments that more closely correspond to only before and after the occlusion.

3.3.1 Selecting a Threshold for Video Segmentation

The proposed automated video segmentation calculation presented in Eq. 3.4 still requires us to pick a threshold ξ. Just as there are different valid ways to segment an image, e.g., segmenting an entire person as one segment or separating body parts and/or articles or clothing, different human observers may have a different interpretations of when a video segment ends and a new one begins. No single segmentation threshold will capture these different conflicting notions of what constitutes a segment, but we can be certain that both an extremely aggressive or an extremely conservative segmentation threshold would be undesirable. An aggressive threshold ($\xi \to 1$) would segment nearly every video into segments of short lengths without capturing longer periods of limited activity within videos.

Figure 3.2: Two frames from the 'lemming' video (left) with the ground truth boxes shown in green, and the portions of the frames where pixels within the union of the two ground truth regions are not masked out (right). The partially masked frames on the right-hand side are used in Eq. 3.4

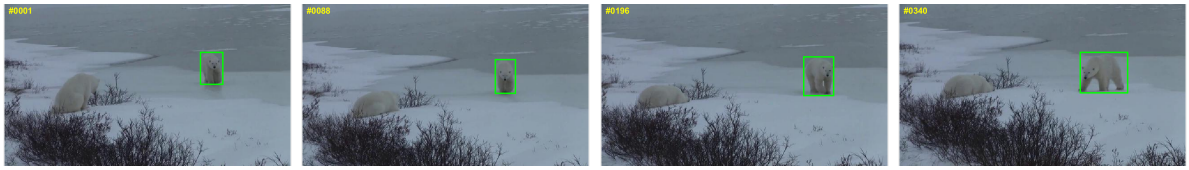
A very conservative threshold ($\xi \to 0$) results in a majority of videos being treated as a single segment; this does not capture the amount of activity in different videos than uniformly short segments. When deciding a threshold, we should expect some videos to have a single segment, while others will have consistently short segments.

We considered thresholds from 0.5 to 0.99, in increments of 0.01. We observed that each change to ξ resulted in approximately a 10% change in the total number of segments; a change of 0.01 in the chosen threshold would probably not be qualitatively noticeable, but as the difference approaches 0.05 and certainly 0.10 there would be a noticeable difference. Ultimately, we choose a threshold of $\xi = 0.9$, after observing video segmentations.

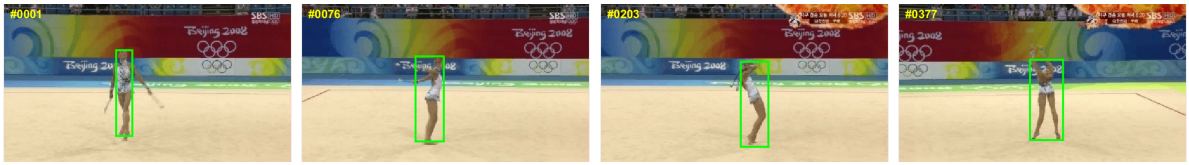
Fig. 3.3 shows frames from videos with a single segment, and videos with among the shortest segments. In particular, we notice that the single segment videos involve very little activity, and very little motion from the original position within the frame. In one video, a gymnast's body is shown deforming in a number of ways, but her motion across the floor is already tracked by a moving camera; overall she does not move within the frame, while staying mostly upright. A tracker would have to actively move off of her, and in this sense, a single segment seems suitable. On the other hand, videos with very short segments tend to have very shaky cameras, or fast moving objects; things that naturally give trackers the most difficulty. We see these trends across the entire set of videos in the benchmarks highlighted in Sec. 3.1.1. While there are some instances where some videos appear to have more or fewer segments than is warranted, we believe there is no easy way to completely prevent such unsatisfactory results. The proposed segmentation is applied equally and without bias across all currently used videos, can be applied to future videos in an identical manner, and produces coherent results that fit both a human's intuition and what trackers find challenging.

.

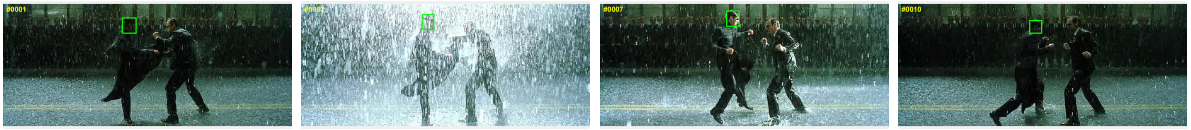
(a) ‘birds1’ sequence - 1 segment, 339 frames



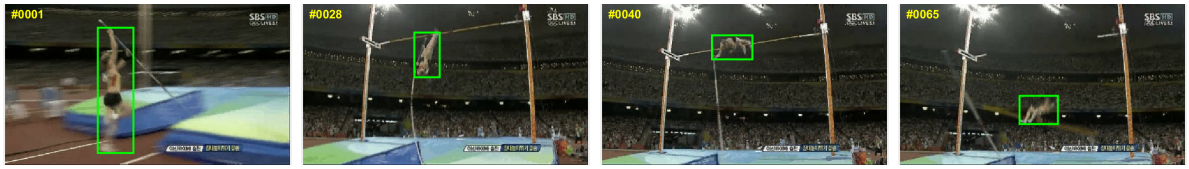
(b) ‘polarbear’ sequence - 1 segment, 371 frames



(c) ‘gym’ sequence - 1 segment, 767 frames



(d) ‘matrix’ sequence - 20 segments, 100 frames



(e) ‘jump’ sequence - 17 segments, 122 frames

(f) ‘human9’ sequence - 60 segments, 305 frames

Figure 3.3: Examples of videos with both a single long segment (a-c) and very short segments (d-f). The ‘human9’ sequence contains considerable camera motion, which may be difficult to observe.

3.4 Tracker Speed

All the discussion above relates to measuring the tracking accuracy for a given tracker. However, because of certain applications that desire real-time tracking or simply require a lot of video processing, tracker speed is often reported more prominently than with other computer vision systems, e.g., single-image object detectors or single-image object recognition systems. With that said, tracker speed is given less significance than the tracker's accuracy. Tracking speed is important in some applications, but accuracy is important in *all* applications. Beyond that, it is often the case that trackers are not optimized for speed; instead they are often built in a manner more closely resembling prototypes in MATLAB. With those caveats in mind, we will still be reporting the FPS for trackers; while the absolute values may not be significant, large relative differences between tracker speeds run in a similar manner are worth highlighting. When computing the FPS, each video in a dataset is given weight. The speeds reported are all for tests run on an Intel(R) Core(TM) i5-4210U CPU using a single core at 1.70 GHz with 8GB RAM.

3.5 Chapter Summary

In this chapter, we reviewed the most commonly used benchmark datasets, which we will also use to experimentally validate our approaches in future chapters. We then discussed how performances on these benchmarks are quantified, and the limittations of those methods. In response to these shortcomings, we propose to measure accuracy and weight portions of videos according to the amount of activity that occurs within them, rather than the sometimes arbitrary length of time before something occurs. In concert with this, we suggest a way to segment videos automatically according to the target being tracked.

Chapter 4

Correlation Filter Design for Visual Tracking

4.1 Introduction

In this chapter, we explore the use of popular CF designs that have been mostly untested for visual tracking applications. As we previously discussed in Ch. 2, there is a wealth of CF work as well as a large number of CFTs, but mostly consisting of trackers built upon the same MOSSE filter or KCF design. The success of different CFs in other localization and detection applications indicates that exploring their application in visual tracking is merited.

We discuss different changes that CFTs have introduced to the basic MOSSE filter design to further meet the challenges of visual tracking, but again emphasize that the fundamental design remains rooted in the ridge regression solution provided by the MOSSE filter.

Based on prior discussion of CF designs in Ch. 2, we explore the use of different CF designs for visual trackers. We compare our methods to the MOSSE and KCF trackers in tests that keep the other tracker components the same. We show that while other CFs may

not fit in the visual trackers as naturally as the CFs explicitly designed for tracking, we can adapt them to the problem setting and even fuse them with native tracking CFs to achieve performance gains, which we validate on the benchmark datasets previously introduced in Ch. 3.

4.2 Prior Correlation Filter Designs for Visual Trackers

Ch. 2 discussed the MOSSE filter and tracker [1] and the KCF tracker, which extended the design of the MOSSE filter to non-linear correlation [2]. Both trackers introduced a new CF design as part of a simple tracker using simple features, no scale estimation, and overall no techniques for specifically addressing any particular challenge in visual tracking data. In that sense, they perfectly represent what tracking-by-detection means; the CF detector is nearly the entire tracking system. Of course, as discussed in Sec. 2.4, a number of CFTs have since been designed that address some of the challenges. Just as new trackers have extended the earliest CFTs to incorporate more powerful features, address scale changes, etc., some trackers have looked to further develop the core MOSSE and/or KCF designs to improve tracking.

When KCF was discussed in Sec. 2.3.2, the Gaussian kernel (Eq. 2.41) was provided as an example of the type of non-linear correlation that may be implemented within the CF. However, choosing how to define the kernel $\mathbf{K}$ is an open-ended problem; the KCF tracker explored the use of Gaussian, polynomial, and linear[1] kernels before choosing the Gaussian kernel. Rather than choose a single kernel beforehand, the MKCF tracker [28] proposes using multiple kernels while learning the appropriate weights for them during tracking, i.e.,

[1]The use of a linear kernel in the KCF tracker makes the tracker nearly equivalent to the MOSSE tracker, except for some small differences in the broader tracker system.

$$\mathbf{K} = \sum_{m=1}^{M} d_m^2 \mathbf{K}_m, \tag{4.1}$$

where $\mathbf{K}_m$ is the mth kernel matrix and d_m^2 is the weight [2] given to $\mathbf{K}_m$, such that $\sum_{m=1}^{M} d_m = 1$. Each individual value of the kernel matrix $k(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{d}^T \mathbf{k}(\mathbf{x}_i, \mathbf{x}_j)$, where $\mathbf{d} = (d_1, d_2, \ldots, d_M)$ and $\mathbf{k}(\mathbf{x}_i, \mathbf{x}_j) = (k_1(\mathbf{x}_i, \mathbf{x}_j), k_2(\mathbf{x}_i, \mathbf{x}_j), \ldots, k_M(\mathbf{x}_i, \mathbf{x}_j))^T$. Because the weights of each kernel $\mathbf{K}_m$ must be learned, learning the filter requires the joint optimization of the weights $\mathbf{d}$ as well as the dual-space coefficients $\boldsymbol{\alpha}$, where

$$\boldsymbol{\alpha} = \left(\sum_{m=1}^{M} d_m^2 \mathbf{K}_m + \lambda \mathbf{I} \right)^2 \mathbf{g}, \tag{4.2}$$

where $\mathbf{g}$ is the desired correlation output. $\mathbf{d}$ and $\boldsymbol{\alpha}$ can be computed analytically when the other is held constant, and alternately solving for $\mathbf{d}$ and $\boldsymbol{\alpha}$ provides close to convergence after a few iterations. Once $\mathbf{d}$ and $\boldsymbol{\alpha}$ are computed, a correlation output for a new image can be computed in a similar manner as KCF (Eq. 2.42).

While MKCF is a general formulation using an arbitrary number of kernels, in practice the authors only implement two separate Gaussian kernels (see Eq. 2.41) for color feature attributes and HOG features. The two kernels have slightly different kernels, but the multi-kernel approach mainly serves as a way for the tracker to learn the weighting between different types of features over the course of tracking. A newer version of the MKCF tracker using CNN features was introduced for the VOT2016 benchmark [45], but details of the exact modifications are sparse.

As previously discussed, computing correlation via the frequency domain will produce aliasing effects caused by circular correlation, unless the signals are sufficiently zero-padded. This procedure is typically not performed in CFTs; instead the support of the CF template is usually larger than the target size, and the training images are windowed to both emphasize the target region and reduce the affects of aliasing. However, nothing in the design of the CF itself addresses the larger filter size that includes background clutter. The Spatially

[2] The weights d_m are squared to ensure negative weights are not computed in the subsequent optimization.

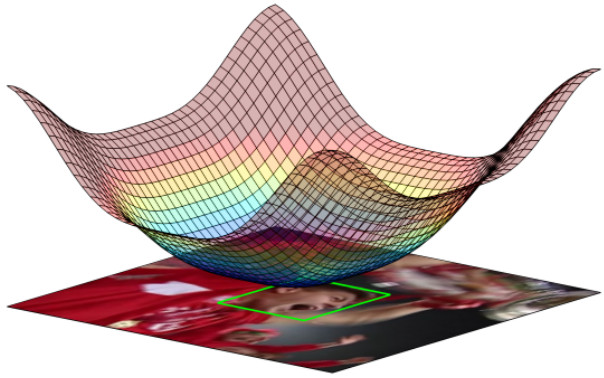


Figure 4.1: Illustration of the spatial regularization weights introduced in SRDCF. The target is the region within the green box, and the entire image shows the full size of the training sample and subsequent CF, which is typical of many trackers. The increased weights on the background portions of the region result in a greater emphasis on the region corresponding to the target. Image taken from [26].

Regularized Discriminative Correlation Filters (SRDCF) [26] adds weights to the MOSSE filter design that emphasize learning from the center of the training image, i.e.,

$$\mathbf{w} = \arg\min_{\mathbf{w}} \frac{1}{N} \sum_{i=1}^{N} \|\mathbf{w} \otimes \mathbf{x}_i - \mathbf{g}_i\|_2^2 + \lambda \|\mathbf{s} \cdot \mathbf{w}\|_2^2 \tag{4.3}$$

where $\mathbf{w}$ is the space-domain filter to be learned, $\mathbf{x}_i$ is the ith training image, $\mathbf{g}$ is the desired correlation output, and $\mathbf{s}$ are the new spatial regularization weights. Fig. 4.1 illustrates how the weights increase at the edges of the filter. The addition of $\mathbf{s}$ makes it hard to solve for the filter in Eq. 4.3 analytically like MOSSE, but it can be efficiently solved in the frequency domain using an iterative method when $\mathbf{s}$ has sparse DFT coefficients. In practice, the authors of SRDCF compute $\hat{\mathbf{s}} = \mathcal{F}\{\mathbf{s}\}$ and retain only approximately 10 DFT coefficients which have significant magnitudes.

The spatial weights emphasize the target region more, and the tracking performance improves with their introduction. The work has similarities to earlier work in zero-aliasing CFs [69], which introduced hard constraints on CF regions that corresponded to the zero-padded regions in the more traditional CF training scheme. The reduced energy in the SRDCF corresponding to background portions of the training images can be observed in Fig. 4.2.

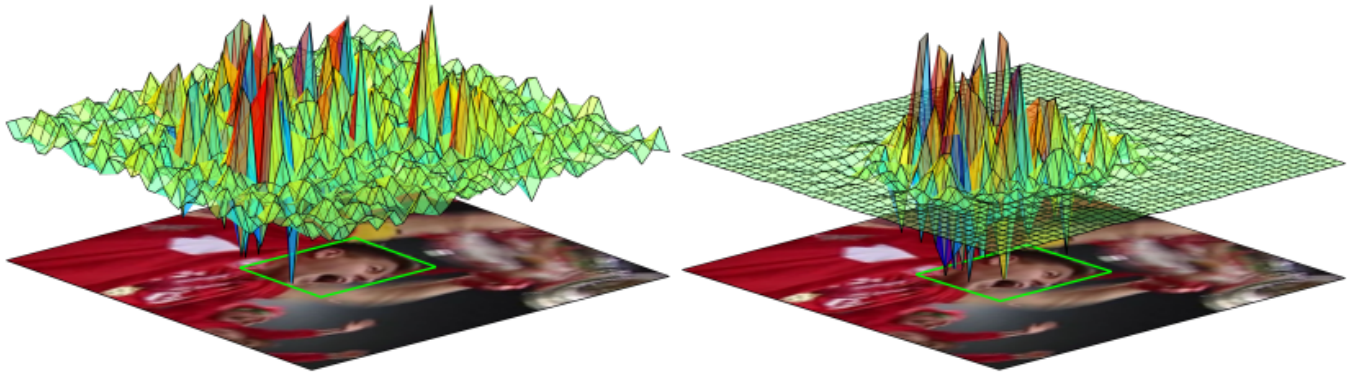


Figure 4.2: Illustration of the effect spatial regularization has on the learned CF. The conventional CF, left, contains a lot of energy in the regions corresponding to the background clutter around the target, while the SRDCF, right, has significantly less energy in the filter corresponding to those regions. Image taken from [26].

MKCF and SRDCF aim to improve the CF by focusing on the kernelized correlation and the training strategy for the filter, respectively, but both are derived from the MOSSE filter's basic design. Liu et al. [70] propose one of the few trackers that use non-MOSSE CFs, though not in the same manner that most CFs are employed. Liu et al. use OTSDF filters to perform visual tracking via what they refer to as correlation filter coding (CFC). Rather than build a single CF and locate the target based on a single correlation output, the CFC builds an ensemble of filters to match either the target or background clutter. Once the filter bank $(\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_N)$ is trained, the filters are applied to the region of interest in the next frame and produce correlation outputs $(\mathbf{g}_1, \mathbf{g}_2, \ldots, \mathbf{g}_N)$. The correlation outputs at a given location become an N dimensional feature vector, and a Naive Bayes classifier is used to determine the final estimate of the target location, based on the N correlation outputs at each possible location in the region of interest. With each new frame, the new target detection replaces the oldest instance of the target in a image bank used to train the OTSDF filters, and the background clutter images in the training set are updated periodically, though not every frame. While the training images are constantly refreshed, the OTSDF filters are still updated according to a linear combination of the previously computed filter and a new filter built from the most recent training set. Because there is

an ensemble of OTSDF filters, each filter is trained from only a subset of the full training image set. Based on numerical experiments, the authors report better performance than the KCF tracker on the OTB50 benchmark.

4.3 Alternative Filters for Visual Tracking

Despite the improvements to CFTs discussed in Sec. 2.4 and specifically to the filter design, as discussed in Sec. 4.2, there appears to be no effort to use a non-MOSSE CF in the same manner that most CFTs employ the filter. Despite other filter designs outperforming MOSSE in other localization tasks [63], their use in visual tracking is unexplored, even as some trackers become more complex and seem to emphasize speed less.

We examine the possible use of three different filters in visual tracking, chosen based on their strong performance in other vision tasks:

1. *OTSDF* [10, 11]: The OTSDF filter, discussed in Sec. 2.2.3, jointly minimizes the average correlation energy and the output noise variance. The tradeoff between the two criteria results in output correlation planes with sharp peaks while still being tolerant to unseen distortions in test images.
2. *UOTSDF* [71]: The Unconstrained Optimal Tradeoff Synthetic Discriminant Function (UOTSDF) filter eliminates the hard peak constraints of the OTSDF filter. Instead, the filter maximizes the average peak height among its training images and minimizes the dissimilarity between correlation outputs from different training images. The UOTSDF filter optimizes these two criteria along with the ACE and ONV criteria that the OTSDF filter already optimizes.
3. *MMCF* [13]: MMCF, discussed in Sec. 2.2.4, combines the localization criterion of CFs like MOSSE with the maximum margin capabilities of SVMs.

The three CFs have not been used in visual trackers, with the exception of the use of

OTSDF filters in CFC [70], as discussed in Sec. 4.2. As we incorporate them into visual trackers, we will compare them to the KCF tracker and the MOSSE tracker[3]. We will primarily use HOG features in our trackers, as KCF previously used.

4.3.1 A Simple Approach to Adapting New Correlation Filters to Tracking

The first approach we take is to simply change the CF used in the CFT. We must point out that the adaptation scheme used by the MOSSE and KCF filters are not directly transferable to the 3 new filters being examined. Recall that the MOSSE and KCF filters adapt to the video over time with a linear interpolation, i.e., $\mathcal{T}_i = (1-\lambda)\mathcal{T}_{i-1} + \lambda\mathcal{T}_{new}$ where λ represents the adaptation rate. The MOSSE filter can use this adaptation scheme because it is designed such that each training image contributes to the computed filter independent of other training images; the λ term only assigns different weights to training images. The KCF filter uses a similar approach, but a OTSDF, UOTSDF, or MMCF filter in a tracking system cannot simply be the linear combination of the previously learned filter and a new filter trained on a single image (from the new frame).

Rather than update an existing filter, we retain the most recent N images to retrain the new CFs. For frames 1 to N, we train our filter on each of the previous target detections, while for frames $N+1$ to the last frame, we train our filter on the N most recent target detections while discarding the older frames. Each of the images are given equal weight. These filters are inserted into the KCF tracker and replace the original KCF filter. The OTSDF and UOTSDF filters also use negative training images; following each detection, background patches adjacent to the target are used to train the filters along with the true class detections. The MMCF filter is not trained with background patches; we found that this method actually performed worse than using a zero-vector as a negative class example,

[3]In practice, the MOSSE tracker is the KCF tracker with linear kernel. This ensures that the other elements of the tracking system are same and optimized for benchmark performance.

Tracker	OTB50	OTB100	VOT2013	VOT2014	VOT2015
MOSSE	0.546	0.472	0.590	0.489	0.342
KCF	0.553	0.475	0.596	0.492	0.340
OTSDF	0.330	0.300	0.412	0.339	0.232
UOTSDF	0.431	0.386	0.490	0.401	0.292
MMCF	0.379	0.353	0.447	0.370	0.249

Table 4.1: ASO on visual tracking benchmarks when using a single CF design for tracking. See Sec. 4.3.1 for details.

which is the choice we make instead of using background clutter.

Table 4.1 shows the ASO when $N = 50$, along with the MOSSE and KCF benchmarks. We can immediately see that all 3 filters perform much worse than the MOSSE and KCF trackers. One contributing factor is that a number of tracks perform poorly from the initial frame; this could be because there is insufficient data to effectively train the filters on the initial frame. We address this in Sec. 4.3.2.

We also tested the different filters with different amounts of training images being retained. Plots showing the tradeoff between accuracy and speed can be seen in Fig. 4.3. We observe that most performance gains are made as the number of training images is increased to 20, with some additional gains as the training size increases to about 50 previous detections in some cases. It is likely that older images are no longer helpful in characterizing the current appearance of the target. In Secs. 4.3.2-5, we use no more than 50 previous frames. We also point out that, as would be expected, the tracking speed drops as more images are retained for training. By comparison, the MOSSE tracker and KCF tracker are much faster, with speeds of roughly 60 and 50 FPS, respectively. While the slower speeds are not optimal, the more important issue to address was the poor accuracy of the newly introduced filters.

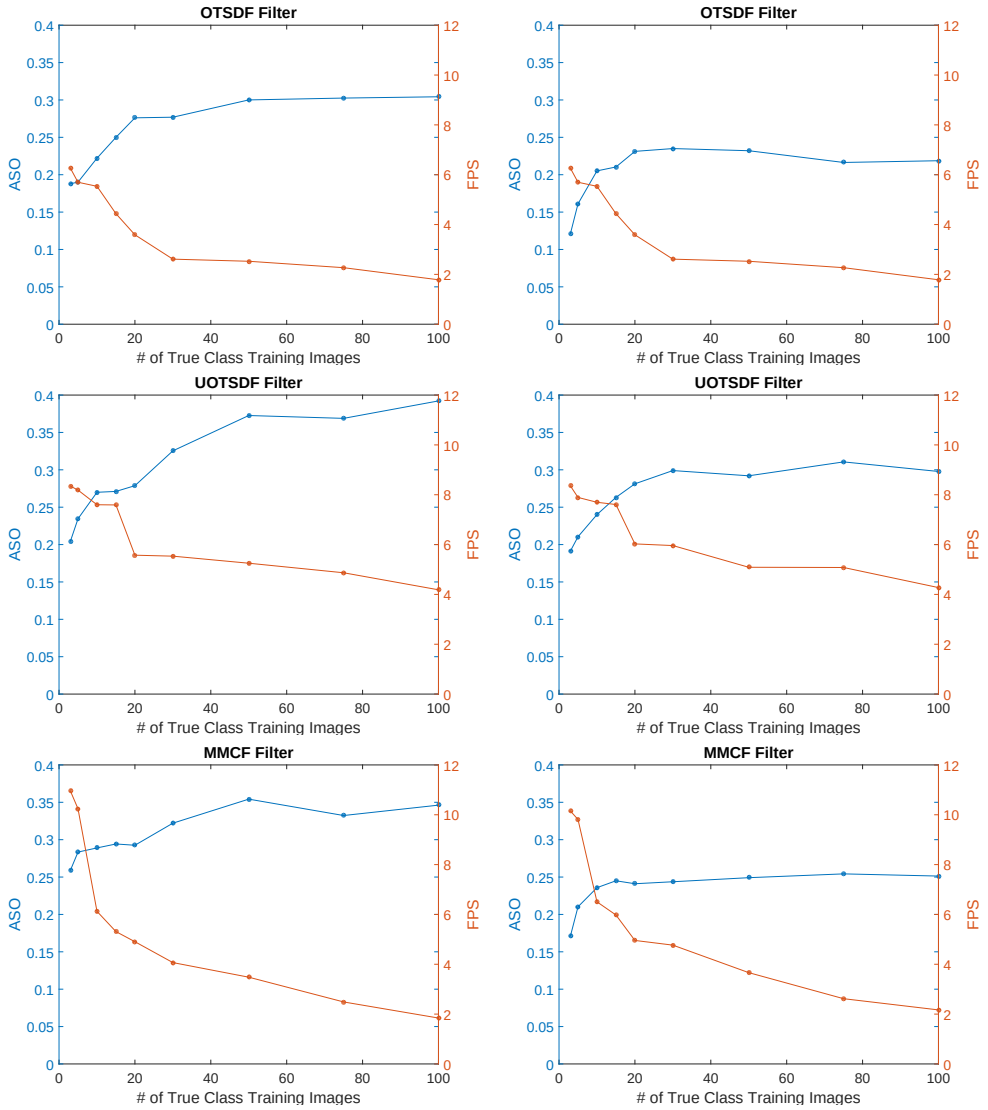


Figure 4.3: ASO (in blue) and FPS (in red) of OTSDF, UOTSDF, and MMCF trackers on OTB100 (left) and VOT2016 (right) when varying amounts of previous detections are retained for training the filter. See Sec. 4.3.1 for details.

Tracker	OTB50	OTB100	VOT2013	VOT2014	VOT2015
MOSSE	0.546	0.472	0.590	0.489	0.342
KCF	0.553	0.475	0.596	0.492	0.340
OTSDF	0.377	0.338	0.506	0.386	0.260
UOTSDF	0.453	0.397	0.524	0.413	0.292
MMCF	0.432	0.395	0.526	0.405	0.262

Table 4.2: ASO on visual tracking benchmarks; trackers using the OTSDF, UOTSDF, and MMCF filters initially use the KCF filter until enough training images are acquired, at which point the KCF filter is not used. See Sec. 4.3.2 for details.

4.3.2 Filter Handoff Approach

We adjust the trackers using the newly introduced factors, based on the performance shown in Sec. 4.3.1. Observing the new trackers fail immediately on certain tracks, it appears that the filters require more training data to learn a robust filter. This is in contrast to the MOSSE and KCF filters, which generally handle early motion just as effectively as motion later in the tracks.

To address this problem, we run our new trackers in two parts. For the first N frames, we employ the KCF filter while retaining the detections. Beginning at frame $N + 1$, we immediately transfer over to using one of the new filters (OTSDF, UOTSDF, MMCF). At this point, the amount of training data should not be an issue. However, examining the results in Table 4.2 show that while there is an improvement, it is marginal, and it is not sufficient to approach the performance of the baseline trackers.

4.3.3 Filter Fusion Approach

Based on the results of using the new filters independently, both with and without an initial run of the KCF filter, a new approach is required. Rather than run the new filters independently, we investigate if they can effectively support the existing KCF filter tracker. Rather than only using the new filters, or running a KCF filter before switching to the new filter, we run both KCF and a new filter and combine the correlation outputs from each.

Fusion of multiple CFs can be performed at a number of levels of the detection stage. We could choose to fuse the filters themselves, the output correlation planes, or the estimated detections. Fusing the filters runs into an immediate problem; the KCF filter is never directly computed, only the dual-space coefficients for the filter (see. Eq. 2.40). Beyond the issue of feasibility, combining disparate filters may just dull the benefits of each independently learned filters. At the other extreme, fusing estimated detections may make sense if the different detectors are entirely different from each other, or if a large number of trackers are used, i.e., enough to determine a dominant mode. With just two CF detectors, there is no compelling reason to allow one bad detection to poison the final result, and no compelling reason to discard all the information from each output correlation plane before fusing the results.

Indeed, the most sensible option appears to be to fuse the output correlation planes. Given two CFs $\mathbf{h}_1$ and $\mathbf{h}_2$, we might normalize the energy in each to control the contribution of each to the final output. However, this is complicated by the fact that the KCF filter is not directly computed. Instead, we normalize the energy in the output correlation planes $\mathbf{g}_{KCF}$ and $\mathbf{g}_{new}$, where $\mathbf{g}_{new}$ is the output correlation plane from one of the 3 new filters being tested. Normalizing the energy of the output correlation planes is akin to computing the peak to correlation energy (PCE) planes of each output. PCE is a measure that scales a peak value according to the total energy in the correlation plane and can help normalize high (or low) peak values in instances where the entire correlation output is high (or low) due to image characteristics such as the illumination. In the case of normalizing two different output correlation planes, normalizing the energy will implicitly give more weight to a single sharp peak in a correlation plane compared to much broader peaks or correlation outputs that produce many peaks. This is a desirable result; a single peak should indicate a higher confidence as to the target's location than a broader peak or an output with multiple peaks.

Beyond normalizing the output correlation planes $\mathbf{g}_{KCF}$ and $\mathbf{g}_{new}$, we do test the fusion

Tracker	OTB50	OTB100	VOT2013	VOT2014	VOT2015
MOSSE	0.728	0.689	0.710	0.617	0.425
KCF	0.740	0.695	0.710	0.617	0.420
OTSDF	0.709	0.655	0.677	0.606	0.402
UOTSDF	0.718	0.690	0.716	0.655	0.460
MMCF	0.721	0.691	0.710	0.616	0.421

Table 4.3: CLE on visual tracking benchmarks; trackers using the OTSDF, UOTSDF, and MMCF filters initially use the KCF filter until enough training images are acquired, at which point the correlation outputs from both filters are used together. See Sec. 4.3.3 for details.

Tracker	OTB50	OTB100	VOT2013	VOT2014	VOT2015
MOSSE	0.513	0.478	0.503	0.416	0.300
KCF	0.519	0.479	0.506	0.418	0.296
OTSDF	0.510	0.463	0.501	0.412	0.297
UOTSDF	0.511	0.479	0.510	0.445	0.316
MMCF	0.518	0.481	0.506	0.419	0.307

Table 4.4: AVO on visual tracking benchmarks; trackers using the OTSDF, UOTSDF, and MMCF filters initially use the KCF filter until enough training images are acquired, at which point the correlation outputs from both filters are used together. See Sec. 4.3.3 for details.

with different weights assigned to each output. Accordingly, our final output plane is a weighted linear combination, i.e.,

$$\mathbf{g}_{fused} = (1-\eta)\mathbf{g}_{KCF} + \eta\mathbf{g}_{new} \tag{4.4}$$

where $0 \leq \eta \leq 1$ is the weight between the two filters. When running the filter fusion trackers, we still run the tracker with an initial period of using only the KCF, in accordance with the improved results shown in Sec. 4.3.2.

The performance of this fusion approach are shown in Tables 4.3, 4.4, and 4.5. The results show mixed evidence that the approach is effective, particularly with the UOTSDF and MMCF filters. Notably, the largest performance gains are on the VOT2015 dataset (and with the UOTSDF filter, the smaller VOT2014 dataset as well). The VOT2015 dataset is the most challenging benchmark; it has the lowest benchmark accuracies and the fusion filters improve these results the most. This is supported by examining the performance

Tracker	OTB50	OTB100	VOT2013	VOT2014	VOT2015
MOSSE	0.546	0.472	0.590	0.489	0.342
KCF	0.553	0.475	0.596	0.492	0.340
OTSDF	0.540	0.464	0.588	0.485	0.343
p-values	–	–	–	–	> 0.05
	–	–	–	–	> 0.05
UOTSDF	0.547	0.477	0.594	0.501	0.361
p-values	> 0.05	< 0.001	> 0.05	0.001	0.001
	–	0.027	–	0.011	< 0.001
MMCF	0.551	0.477	0.595	0.493	0.351
p-values	< 0.001	< 0.001	< 0.001	< 0.001	< 0.001
	–	0.015	–	> 0.05	< 0.001

Table 4.5: ASO on visual tracking benchmarks; trackers using the OTSDF, UOTSDF, and MMCF filters initially use the KCF filter until enough training images are acquired, at which point the correlation outputs from both filters are used together. See Sec. 4.3.3 for details.

on a per-track basis, illustrated in Fig. 4.4. The performance on both the OTB100 and VOT2015 datasets are shown, and the contrast between them, as well as the contrast of the improvements from using the fusion filter approach, are noticeable. The OTB100 benchmark is easier overall, there are more tracks (100 in OTB100 vs. 60 in VOT2015), yet fewer tracks that KCF has an $ASO < 0.2$, indicating complete (or nearly complete) track failure. These tracks where $ASO_{KCF} < 0.2$ are the tracks that the fusion filters have the most success with. When the KCF filter performs better, the fusion filter more often produces the same result, with some sporadic successes and failures. Because the fusion filters most often improve the worst KCF tracks, it is not surprising that the biggest improvements are seen on the VOT2015 dataset. Examples of improved tracks are shown in Fig 4.5, but there is no single characteristic that consistently appears in the tracks that have the biggest improvements with the fusion filters.

To examine the trend of fusion filters improving the worst KCF tracks, we can examine the fusion weighting parameter η and the actual fusion of the output correlation planes $\mathbf{g}_{KCF}$ and $\mathbf{g}_{new}$. A fusion weight $\eta = 0.5$ would correspond to equal weight given to KCF

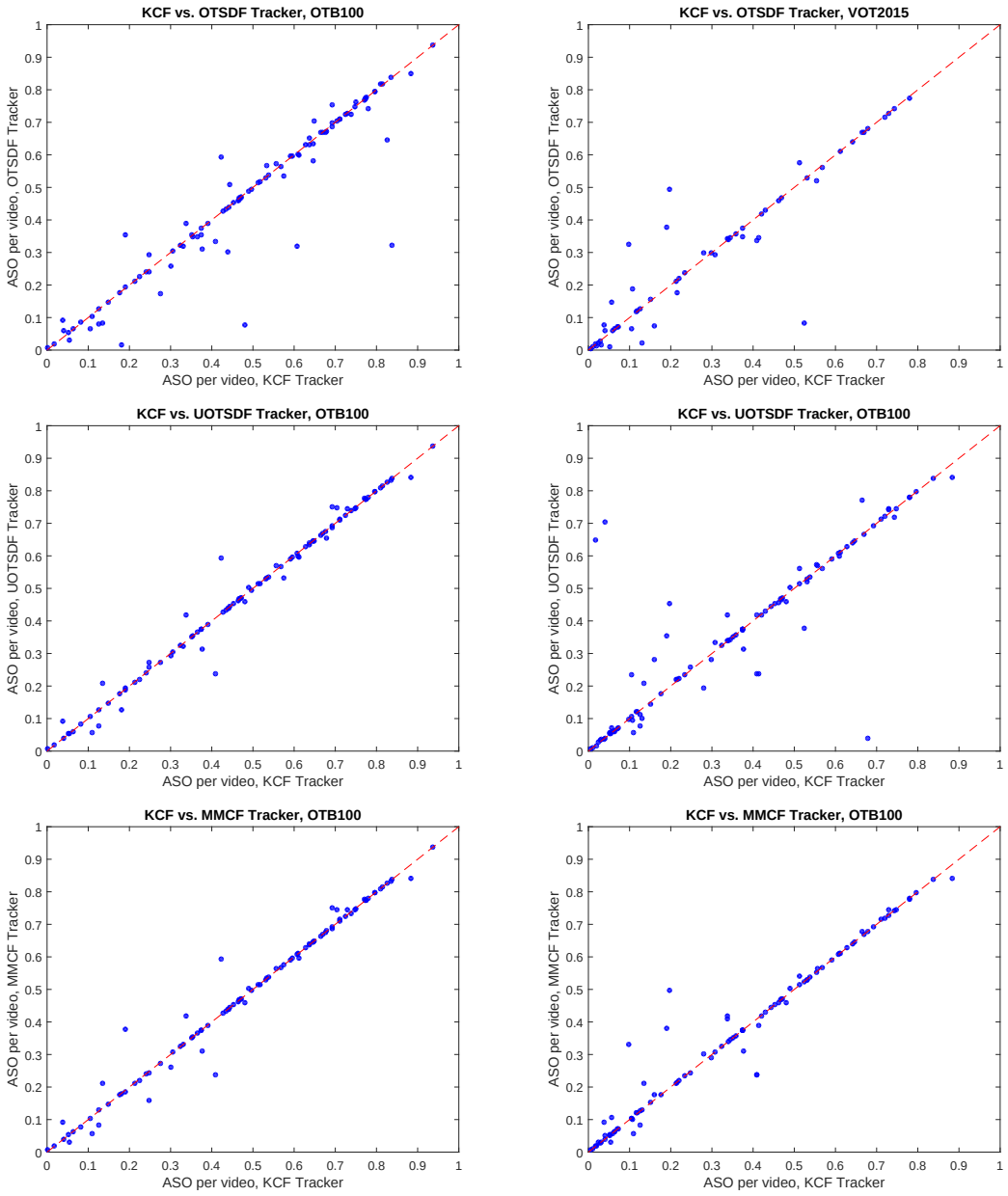


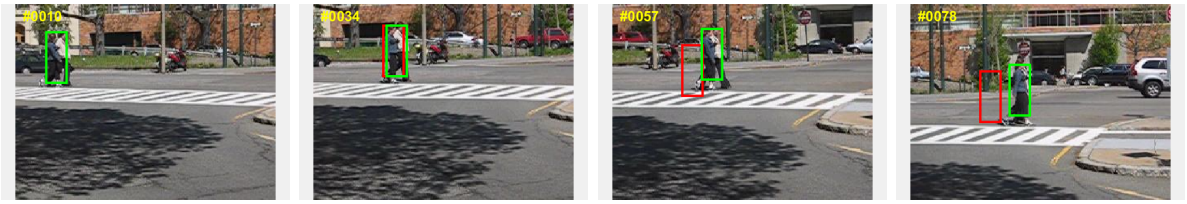
Figure 4.4: ASOs for individual tracks for both the KCF tracker and the fusion trackers on the OTB100 (left) and VOT2015 (right) datasets. We can observe that the new filters, particularly UOTSDF, are most effective at improving the tracks that KCF performs worst on. See Sec. 4.3.3 for details.

(a) ‘blurbody’ sequence, KCF-OTSDF fusion tracker

(b) ‘graduate’ sequence, KCF-OTSDF fusion tracker

(c) ‘gymnastics4’ sequence, KCF-UOTSDF fusion tracker

(d) ‘iceskater1’ sequence, KCF-UOTSDF fusion tracker



(e) ‘couple’ example, KCF-MMCF fusion tracker

(f) ‘mmcf’ sequence, KCF-MMCF fusion tracker

Figure 4.5: Examples of new filters fused with the KCF filter improving the performance of a single KCF filter. Green bounding boxes indicate the fusion trackers, while the red bounding boxes indicate the KCF tracker. See Sec. 4.3.3 for details.

and the new filter, however the best results shown in Table 4.5 use a weight $0.01 \leq \eta \leq 0.1$, indicating that KCF is given much more weight than the newly introduced filter. Within the range of $0.01 \leq \eta \leq 0.1$, there are small fluctuations in performance, but as η approaches 0.5, the performance typically drops noticeably. Consider that the results discussed in Sec. 4.3.2 are equivalent to $\eta = 1$, and it is not surprising that the best performance is when the KCF filter is given more emphasis. Still, allowing the new filter to have a small weight allows the new filter to influence the tracker when the KCF filter outputs the least confident results. A strong KCF output correlation peak will dominate the fusion when $(1 - \eta) >> \eta$, but when no strong peak is present, e.g., in a track the KCF tracker struggles with, the new filter will have an opportunity to correct the output from the KCF filter. Based on our results, it is the MMCF and particularly the UOTSDF filter that performs this task well, while the OTSDF filter does not appreciably help in this regard.

4.3.4 Time-Dependent Image Weighting in Correlation Filter Training Sets

The previously discussed approaches to using new filter designs in trackers all use the most recent N frames to compute the filter. However, this does not align well with the preferred adaptation scheme used by MOSSE and KCF, and it does not fit our intuition that the most recent instances of the target should carry more weight than older appearance information that may be outdated.

The MOSSE and KCF filters use an update scheme $\mathcal{T}_i = (1 - \lambda)\mathcal{T}_{i-1} + \lambda\mathcal{T}_{new}$, and thus the weight given to a new target detection decays exponentially at a rate of $(1-\lambda)$ per frame, while never being entirely discarded. Completely retraining the OTSDF, UOTSDF, and MMCF filters would mean that retaining all images will cause the tracker to progressively slow down over the course of a video as the training set increases, so we keep only the most recent N images. For the N most recent training images, we introduce weights that decay

linearly, i.e., $\mathbf{m} = [\frac{1}{N}, \frac{2}{N}, \ldots, \frac{N}{N}]$.

Depending on the filter design being used, the integration of the weights are slightly different. For the OTSDF and UOTSDF filters, the weights determine the normalized energy of a given training sample; when computing these filters, the energy of each sample is normally fixed to 1 (or some constant value); instead we normalize the energy of training sample $\mathbf{x}_k$ to equal the weight $\mathbf{m}_k$. This weighting is then implicitly carried through the entire computing of the filter, e.g., we compute a weighted PSD. For the MMCF filter, the weights modify the class labels c_k, shown in Eq. 2.21, such that $c_{k_{weighted}} = \mathbf{m}_k c_k$. The class labels c_k impose the margin inequality constraints, and reducing this value over time relaxes the margin constraint required for older training images. An existing solver for this modified SVM formulation is available [72].

We apply these changes to the new CF designs and incorporate them into the fusion filter trackers, discussed in Sec. 4.3.3. The quantitative results from this approach can be seen in Table 4.6. Despite the intuition motivating the added training sample weights, their introduction does not significantly impact the performance. In some cases, such as the OTSDF and MMCF filters on the VOT2015 benchmark, there are improvements, while most often the results are very similar or in some cases, slightly worse, such as the UOTSDF filter on the VOT2014 dataset. Qualitative observation also does not illustrate any clear situations that one approach is regularly more effective than the other. Without any clear advantage, either overall or in the presence of particular challenges, it is unclear whether a recommendation to construct weighted fusion filters, rather than unweighted fusion filters, is warranted.

4.3.5 The Impact of Feature Descriptors on Filter Design

The preceding discussion of filter design for tracking in Secs. 4.3.1-4 all use HOG feature descriptors to characterize training samples. The choice of HOG features is in some sense

Tracker	OTB50	OTB100	VOT2013	VOT2014	VOT2015
MOSSE	0.546	0.472	0.590	0.489	0.342
KCF	0.553	0.475	0.596	0.492	0.340
OTSDF	0.545	0.466	0.588	0.485	0.353
p-values	–	–	–	–	< 0.001
	–	–	–	–	< 0.001
UOTSDF	0.548	0.477	0.594	0.485	0.361
p-values	0.003	< 0.001	0.047	–	< 0.001
	–	0.026	–	–	< 0.001
MMCF	0.552	0.477	0.596	0.493	0.356
p-values	< 0.001	< 0.001	< 0.001	< 0.001	0.004
	–	0.015	–	> 0.05	0.001

Table 4.6: ASO on visual tracking benchmarks; trackers using the OTSDF, UOTSDF, and MMCF filters initially use the KCF filter until enough training images are acquired, at which point the correlation outputs from both filters are used together. The training images for the OTSDF, UOTSDF, and MMCF filters are weighted such that the most recent frames are given more weight. See Sec. 4.3.4 for details.

Kernel	Features	CLE
Linear	Raw pixels	0.451
Gaussian		0.560
Linear	HOG	0.728
Gaussian		0.732

Table 4.7: Performance of KCF on OTB50, as reported in [2]. The use of a linear kernel is equivalent to using the MOSSE filter.

a simple one; previous works have shown its effective combination of being both fast and effective in tracking (see Sec. 2.4.1). However, the use of one feature descriptor instead of an alternative does not simply have an additive effect on the accuracy of the tracker; previous results have shown this without highlighting this fact.

Prior to the publication of the KCF tracker [2], the authors published the preceding CSK tracker [3]. The CSK tracker introduced the kernelized version of the MOSSE filter, albeit in a different form. The experimental results were limited to using raw pixel as feature descriptors on a small number of tracks, and the CSK tracker outperformed the MOSSE tracker (and 2 other trackers) significantly. The following work reformulated the CSK

tracker into what is widely referred to as the KCF (or sometimes DCF, for Discriminative Correlation Filter) tracker. The KCF tracker was formulated in a manner that was more clearly derived from the MOSSE filter, detailed different possible kernelizations for the filter, and additionally introduced the use of multi-channel, i.e., HOG, features. The results on the OTB50 benchmark showed an improvement when introducing the HOG features, and even further improvement over the use of a linear kernel, an equivalent to the MOSSE filter. However, the results also showed that using a linear kernel was almost as effective as the Gaussian kernel once the HOG features were introduced. The results from [2] are shown in Table 4.7.

The results reported in [2] are reflected in our experiments on a wider set of benchmarks, which includes results from Sec. 4.3.3 as well as results when using just raw pixels, i.e., grayscale intensities, as our feature descriptors. The results of the MOSSE tracker, KCF tracker, and OTSDF, UOTSDF, and MMCF fusion filters are shown in Table 4.8. These results reflect the trend originally shown in [2]; the MOSSE tracker performs significantly worse when pixel intensity features are used, but benefits the most from using HOG features. The other 4 filters all show roughly an equal performance gain with the introduction of HOG features.

While the results in Table 4.8 affirm what was previously shown when KCF was introduced (but not discussed at length); the introduction of a more powerful image descriptor, in this case HOG, appears to nullify the benefits of kernelization, as with KCF, and perhaps the filter design itself. While the introduction of new filter designs to tracking in Secs. 4.3.1-2 could be attributed to a mismatch in the filter design and the problem setting (the filters requiring more training data, the tracking problem being a data starved scenario), even the fusion filters provide a benefit on the most challenging tracks for KCF, but do not provide much benefit above what the KCF or MOSSE trackers are able to achieve when they do not fail entirely, provided they are using HOG features. This is contrary to

Tracker		OTB50	OTB100	VOT2013	VOT2014	VOT2015
MOSSE	Grayscale	0.295	0.288	0.367	0.265	0.185
	HOG	0.546	0.472	0.590	0.489	0.342
	Diff.	+0.251	+0.184	+0.224	+0.224	+0.157
KCF	Grayscale	0.417	0.389	0.440	0.328	0.271
	HOG	0.553	0.475	0.596	0.492	0.340
	Diff.	+0.135	+0.086	+0.156	+0.164	+0.069
OTSDF	Grayscale	0.409	0.380	0.497	0.326	0.275
	HOG	0.545	0.466	0.588	0.485	0.353
	Diff.	+0.136	+0.086	+0.091	+0.159	+0.078
UOTSDF	Grayscale	0.431	0.394	0.446	0.344	0.308
	HOG	0.548	0.477	0.594	0.485	0.361
	Diff.	+0.117	+0.083	+0.149	+0.140	+0.053
MMCF	Grayscale	0.414	0.373	0.481	0.375	0.278
	Hog	0.552	0.477	0.596	0.493	0.356
	Diff.	+0.138	+0.104	+0.115	+0.118	+0.078

Table 4.8: Tracker performance using either grayscale pixel intensity features or HOG features. "OTSDF," "UOTSDF," and "MMCF" refer to the fusion filters introduced in Sec. 4.3.3.

earlier comparisons of different CF designs in ATR [63], but those results largely precede the introduction of multi-channel CFs. It is possible that the richer feature descriptors that HOG provides negates the difference between MOSSE and more powerful filters, while the difference between filters is more distinguishable when they are designed with weaker features, e.g., raw pixel intensities. However, without revisiting localization and recognition tasks akin to ATR, it is difficult to generalize the trends that appear in visual tracking.

4.4 Chapter Summary

In this chapter we discussed different approaches to improving the CF design that is at the core of CFTs. We drew attention to the fact that there was almost no evidence that CF designs outside of the MOSSE filter and improved versions of the MOSSE filter appeared to have been adapted to the task of visual tracking. We explored the use of three filters

– the OTSDF, UOTSDF, and MMCF filters – in visual tracking, based on their strong performance in other localization and recognition tasks. Tracking performance using only these new filters proved to be worse than the performance of the MOSSE tracker and its kernelized variant, the KCF tracker. However, fusing these three filters with the KCF tracker showed that there was utility in incorporating these alternative CF designs into CFTs; the new filters were able to improve performance on some of the most challenging tracks for the KCF tracker. This trend was most clear on the VOT2015 dataset, the most recent and most challenging benchmark dataset included in our tests.

We also examined the shared and sometimes redundant role that more powerful CF designs and more powerful feature descriptors have in tracking. We examined the results previously published that indicated that while a more sophisticated CF design brings a big performance gain in visual tracking, the introduction of HOG features largely negates the previously observed improvement in performance (while doing better than raw pixel intensity features in both cases). We showed these trends carry over to larger datasets and can be observed with the newly tested OTSDF, UOTSDF, and MMCF filters when applied to visual tracking.

Chapter 5

Occlusion Detection and Estimation for Improved Visual Tracking

5.1 Introduction

In this chapter, we discuss one of the most challenging factors in visual tracking – occlusion – and introduce a technique for handling occlusion that can be inserted into a broad range of existing trackers. When a target being tracked is occluded by another object in the scene, two bad things can happen. The first is that the tracker may immediately follow an occluding object and lose the target as the occluding object pulls the tracker away from the target. The second problem is that during a period of occlusion the tracker may learn the appearance of the occluding object such that when the target reappears, the tracker has adapted to follow the occlusion instead of the target. This can occur if the target moves behind a stationary object for a period of time. The challenge posed by occlusion for model-free visual trackers stems from the fact that successful tracking requires a constant adaptation to a target's changing appearance. The lack of prior target knowledge makes it challenging to distinguish between a new target appearance and an obscured target.

We introduce Hue-Based Occlusion Estimation (HuBOE), a lightweight occlusion es-

timator that uses color information to determine if the target is occluded or not. When the target is determined to be occluded, we are able to intervene and halt the normal model updates that occur in most CFTs. We demonstrate HuBOE's broad applicability to improve the performance of CFTs with experimental results on the OTB and VOT datasets.

5.2 Tracking Approaches for Handling Occlusion

The simplest trackers do not explicitly address occlusion. Trackers can benefit significantly from having an awareness that the target is likely hidden from view, both for estimating the current target state, and also for maintaining the best possible target models for the remainder of the video. Most CFTs have a simple method for updating the target model in each new frame; recall from Sec. 2.3.3 the general update scheme many CFTs [1–3, 19, 22, 23, 26, 32] employ:

$$\mathcal{T}_i = (1 - \lambda)\mathcal{T}_{i-1} + \lambda\mathcal{T}_{new}, \tag{5.1}$$

where $\mathcal{T}_{i-1}$ contains the previously used CF (or the data necessary to compute the CF), $\mathcal{T}_{new}$ incorporates the information from the new frame, and $0 \leq \lambda \leq 1$ is the adaptation rate. Examples of this model update for specific CFTs have been previously shown in Eqs. 2.37 and 2.43. When $\lambda = 1$ the CFT has no memory of previous frames, and when $\lambda = 0$, the CF will use only information from the initial frame for the entire track. Both extremes have clear problems in a range of scenarios; memory is required for robustness to noise or frames where the target is not visible, and adaptation is required for robustness to the changing appearance of the visible target. Instead of either extreme, trackers are set to a λ (usually closer to 0) that balances these factors. Still, leaving λ at a constant value for all frames of all videos is not optimal; when the target is occluded, there is no useful data to update the CF with, and clearly we want $\lambda = 0$ for that individual frame. To be able to turn off the adaptation rate, the tracker must accurately determine if the target is

occluded. This is a significant challenge and one of the fundamental problems of model-free tracking, because the nature of model-free tracking makes distinguishing occluding objects from unseen target appearances difficult.

The earliest CFT, the MOSSE tracker, used PSR (Eq. 2.2) to measure peak quality. Bolme et al. report that "ranges between 20.0 and 60.0 ... indicate very strong peaks ... when PSR drops to around 7.0 it is an indication that the object is occluded or tracking has failed [1]." However, the test dataset includes only 7 fairly simple videos. The introduction of the KCF tracker does not include any peak quality measure, possibly because the larger, more challenging OTB50 dataset made using a simple PSR threshold less feasible [2, 3]. Still, the approach used by MOSSE highlights two themes: in some manner the quality of the match much be discerned, and detecting occlusions and detecting tracking failures are often indistinguishable. Other approaches discussed below support these trends.

The MUlti-Store Tracker (MUSTer) [25] tracks targets through a combination of a short-term CF tracker and long-term keypoint matching with Scale-Invariant Feature Transform (SIFT) descriptors [73]. During the course of tracking, MUSTer tracks and stores keypoints from both the target and the background. Once the target region for a given frame is estimated, the target region is analyzed according to the number of matched target and background keypoints within the region. Empirically, it was determined that if more background keypoints than target keypoints are present within the estimated region, then the object is likely occluded. When occlusion is detected, the keypoint sets and CF are not updated.

The Collaborative Correlation Tracker (CCT) [24] also uses long- and short-term components. The short-term KCF tracker searches for the target in a local region based on the previous detection, while a long-term detector searches the entire new frame and finds 10 candidate locations without using the prior location. Ideally, one of the 10 regions will

overlap with the KCF estimate, in which case the KCF estimate is accepted. When there is no agreement (less than 5% overlap) between the tracker and all 10 candidates regions, one of two actions are performed. They propose simply reducing the learning rate of the KCF by a factor of 10, or they propose running the KCF in the 10 candidate regions, and then choosing the high correlation value out of all 10 regions as the new estimated location. The first option of reducing the learning rate aligns more closely with handling occluded targets, while the second option is more of a pure redetection approach, which may be applicable to general cases of the tracker losing or drifting from the target. This may actually be a problem during occlusion (when redetection is likely to pull the tracker off the target), but it is addressed by biasing redetections to favor windows closer to the previous location. The authors of [24] show experimental results for both the approaches of reducing the learning rate and redetecting the target, and show that the redetection approach is slightly better.

Walsh and Medeiros [74] developed a failure detection system to augment an existing CFT. Following the application of the CF in the region of interest to localize the target, the average correlation and the entropy of the correlation plane are computed and compared to a weighted average of the average correlations and correlation plane entropies in the last N frames. When average correlation and/or the entropy are below certain thresholds, the CF model update may be stopped, or the tracker may be prevented from estimating a new bounding box altogether. Additionally, as the uncertainty of the target localization increases, the search window in future frames becomes larger. This approach is used to augment an existing CFT [40] and shows a small improvement, particularly on small targets or targets that leave the field of view, but very little improvement on videos with target occlusions.

Other non-CF trackers have also attempted to address the challenges of occlusion, but the mechanisms for detecting and handling occlusion are often similar. Nguyen et al. [75, 76] employ a Kalman filter tracker, and additionally monitors which pixels in the target region

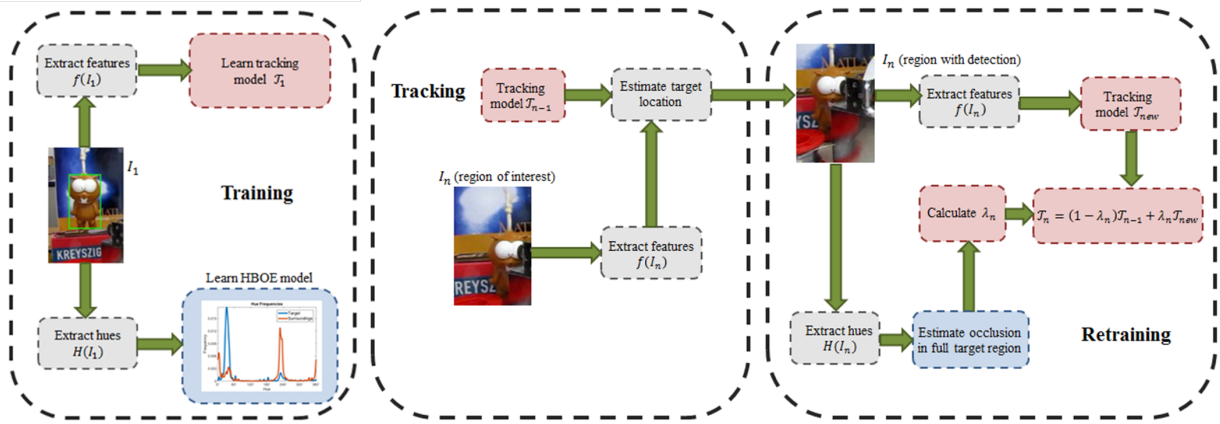
are outliers that have significant appearance changes. The tracker assumes that the target is occluded when too many pixels within the target region become outliers, and stops model updates when this is the case. The model updates resume when the number of outliers drops below the threshold or the length of the occlusion reaches 20 frames. Pernici and Del Bimbo proposed a tracker that detects the target based on distinctive SIFT features relative to the part's surroundings [77]. Similar to MUSTer, when the target is detected, the number of SIFT keypoints within the target region that belong to the surroundings are counted, and if too many keypoints belonging to the surroundings are within the target bounding box, the model for both the target and the surroundings are not updated.

Finally, we recall the parts-based CFTs previously discussed in Sec. 2.4.3. As mentioned, parts-based approaches naturally lend themselves to handling partial occlusions, in which the unoccluded parts provide a reliable detection, and the occluded parts can be identified by anomalous outputs relative to the other parts.

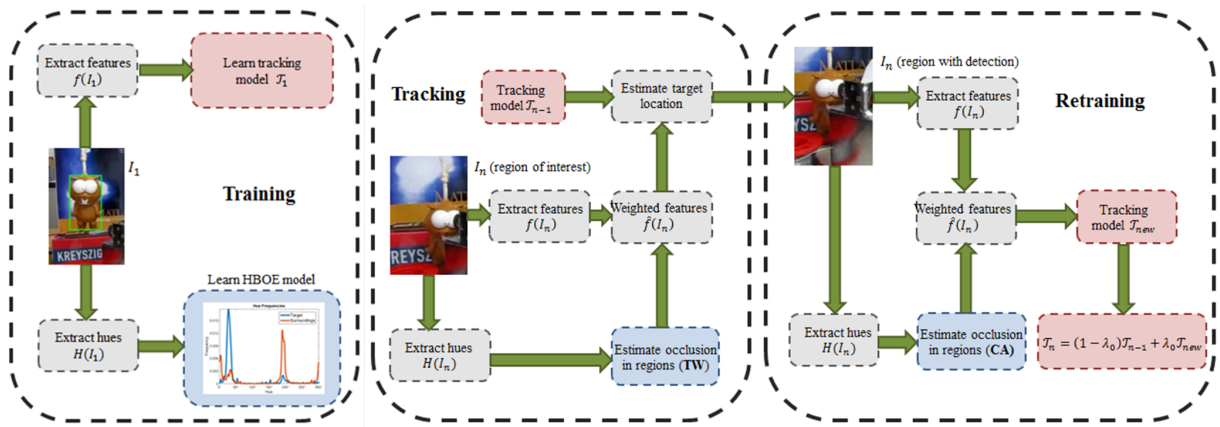
5.3 Hue-Based Occlusion Estimation

Our proposed occlusion detection augments existing trackers by characterizing the color of a target and its surroundings in the first frame [78]. Unlike other methods for detecting occlusion, the proposed approach works independent of the tracking model used, and can augment many existing trackers. Based on the colors visible in the first frame, HuBOE makes inferences about possible occlusions that help control the tracker's adaptation rate and also help estimate the target's movements.

The work is derived from one main assumption: over a short period of time, the color of a target will not change much. When a target's colors are distinct from the target's surroundings, color features can be a useful tool for tracking. While some trackers [22, 23, 25, 32] successfully use color information to find the target, HuBOE identifies regions in a frame that are likely *not* the target. This can indicate when the target is



(a) Tracking-by-detection with full-target HuBOE



(b) Tracking-by-detection with parts-based HuBOE

Figure 5.1: Outline of visual tracking and how full-target and parts-based HuBOE are incorporated into a tracking system. Gray boxes represent processing the current frame, red boxes indicate use of the tracking algorithm, and blue boxes represent when HuBOE is used in tracking. In the parts-based HuBOE, "CA" refers to controlling the adaptation rate (Sec. 5.3.2), and "TW" refers to feature weighting (Sec. 5.3.3).

occluded and thus allow us to change the filter adaptation rate or the new frame's feature representation accordingly. In this section we discuss how we measure the color of the target and how it can be applied to detecting occlusion on either the full target region or in a parts-based approach. An overview of how both full-target and parts-based HuBOE fits into an existing tracker can be seen in Fig. 5.1.

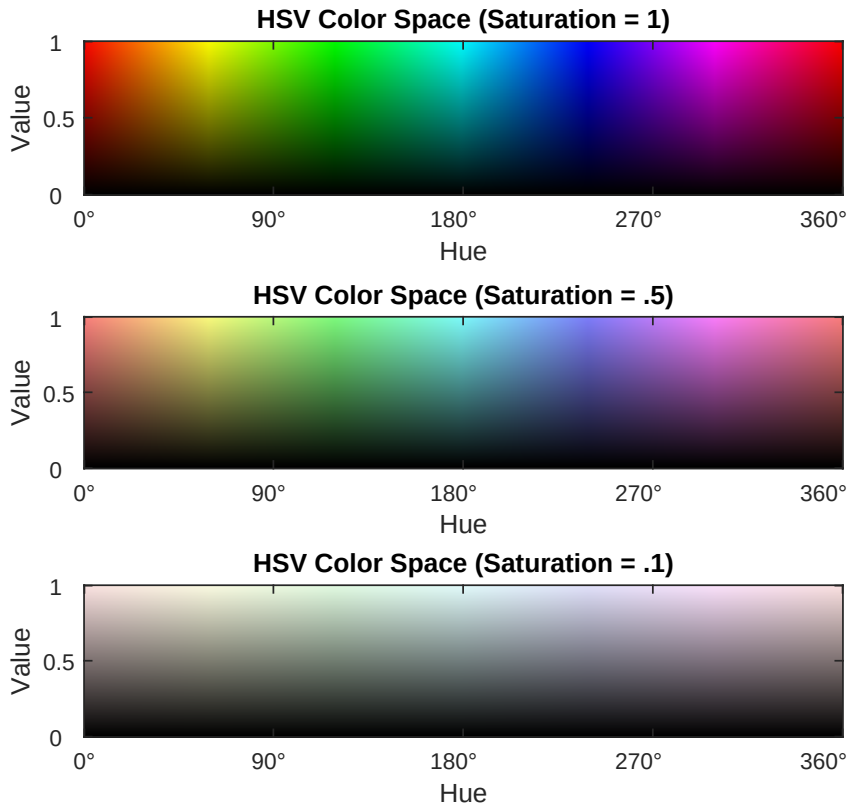


Figure 5.2: 2-D representations of HSV color space at different saturation levels. We can observe the that value and saturation help define the brightness of a pixel and would be affected by illumination changes, while the hue is closely tied to human perception of different colors from red, yellow, green, blue, and then red again as the hue values wrap circularly, and would be less affected by changes in lighting intensity.

5.3.1 Characterizing a Target's Color

One of the many challenges present in visual tracking is illumination variation. Illumination changes can dramatically affect RGB values in color images. Our goal is to capture the color of a scene regardless of the illumination in the first frame. Since RGB values can be affected by lighting, we choose to both reduce the dimensionality and increase the illumination invariance of our color representation by converting RGB values to hue values, according to the standard conversion from RGB color space to the HSV (hue, saturation, and value) color space, which is illustrated in Fig. 5.2. The saturation and value of each pixel are discarded.

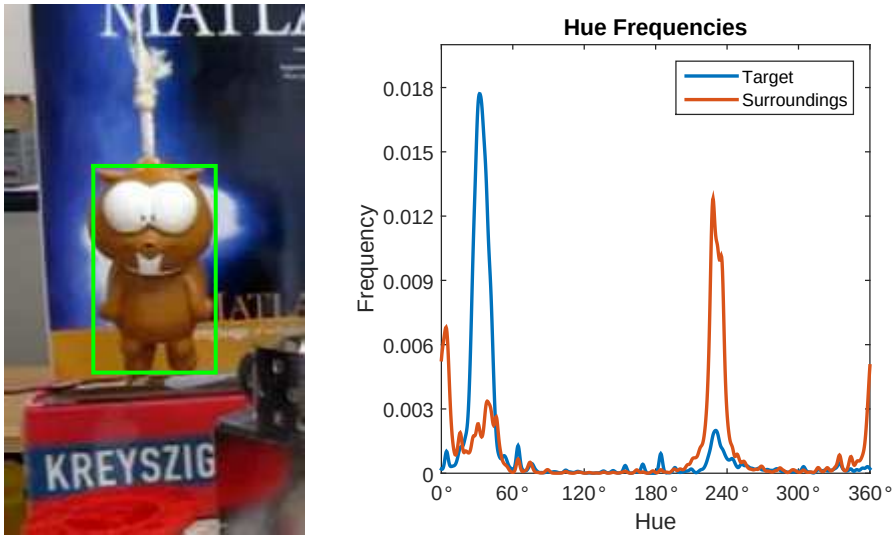


Figure 5.3: Portion of initial frame of 'lemming' track with target shown (left), and distribution of the hues found in the initial frame (right). The two largest peaks correspond to brown and blue hues.

After converting the initial RGB image to hues, we examine the hues both inside the initial bounding box and the surrounding area. Instead of defining the size of the surrounding area of interest, we use the same surrounding area that the CFT being used defines it; it is the region centered at the central point of the target (but not including the target), with a width and height usually between 2x and 3x larger than the estimated target size, depending on the particular CFT implementation. We perform kernel density estimation to estimate $P(H|\text{target})$ and $P(H|\text{surr})$, where H represents a particular hue, from these regions in the first frame. With these distributions, we can compute the log-likelihood of a hue belonging to the background as

$$\mathcal{L}(\text{surr}|H) = \log\left(\frac{P(H|\text{surr}) + \varepsilon}{P(H|\text{target}) + \varepsilon}\right) \tag{5.2}$$

where $\varepsilon = 10^{-4}$ and is included to address cases when the estimated $P(H|\text{target})$ or $P(H|\text{surr})$ equals zero, which can happen due to the small sample from which both distributions are estimated. An example of these estimated distributions on the first frame of a video is shown in Fig. 5.3.

Two assumptions are made at this point. First, we assume that $P(H|\text{target})$ will not

change over time. Second, we assume that $P(H|\text{occlusion}) \approx P(H|\text{surroundings})$. In other words, if we observe that $\mathcal{L}(\text{surroundings}|H)$ is large within the estimated target area, HuBOE assumes that the target has not changed, but rather that some object from the surroundings is now occluding the object where the likelihood value is large.

5.3.2 Full-Target Hue-Based Occlusion Estimation

As discussed in Sec. 5.2, an important challenge of online visual tracking is controlling and accurately tuning the adaptation rate λ. Following the tracker producing a target location estimate $\mathbf{b}'_k = [x'_k, y'_k, w'_k, h'_k]$, we decide if the target is occluded or not by first averaging the hue likelihoods within the target:

$$OS_{raw}(n) = \frac{\sum_{p \in \mathbf{R}'_k} \mathcal{L}\left(\text{surr}|H(p)\right)}{|\mathbf{R}'_k|} \tag{5.3}$$

where $H(p)$ is the hue of pixel p from within the region $\mathbf{R}'_k$ defined by $\mathbf{b}'_k$, and $|\mathbf{R}'_k|$ represents the number of pixels within the bounding box. We denote this average the *raw occlusion score*. We recognize that different video sequences and targets will have different color characteristics, so the same raw occlusion score may have different meanings across different videos. From this, we recognize that decisions based on a single score threshold across different sequences using the raw scores may not perform well. Instead, we approximately normalize the scores by treating the initial (ground truth) frame's occlusion score as a bias term, such that

$$OS(n) = OS_{raw}(n) - OS_{raw}(1) \tag{5.4}$$

This normalized occlusion score has a more consistent interpretation across different videos, and approximately measures how much the nth frame differs from the initial (unoccluded) frame in the sequence, as $OS(0) \equiv 0$. An example of occlusion scores changing in the presence of occlusion is shown in Fig. 5.4 (with initial frame shown in Fig. 5.3).

Once we have computed $OS(n)$, we have to make the decision to actually change the

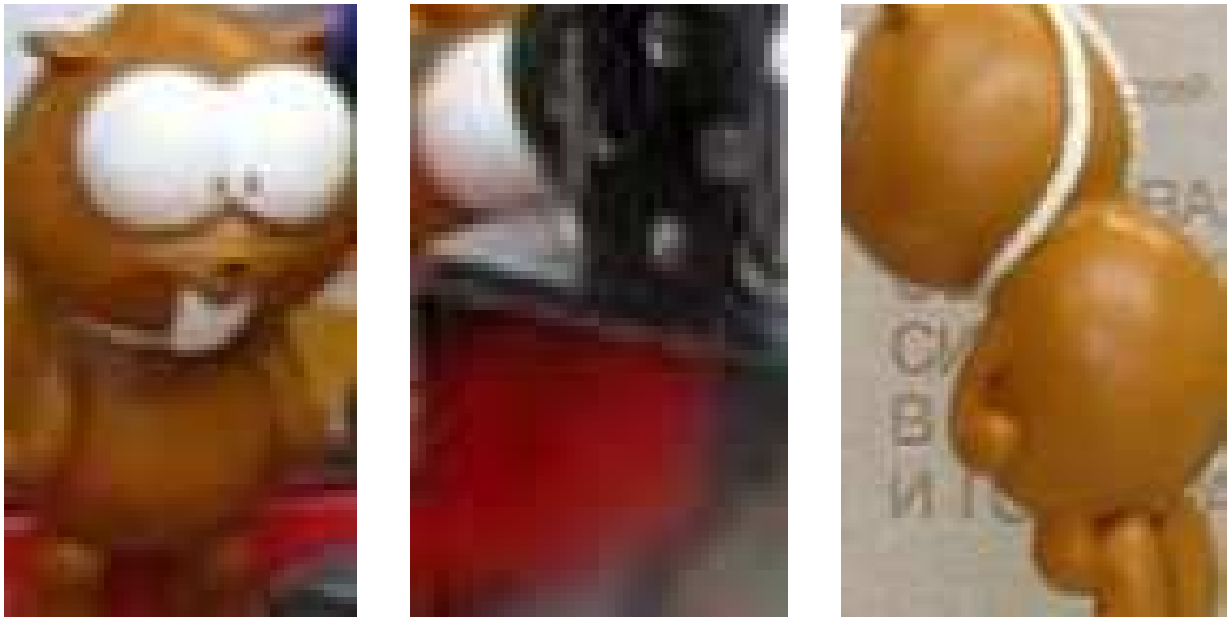


Figure 5.4: Target regions based on ground truth labels in three 'lemming' track frames. Frame 296 (left) has $OS = 0.55$, while the occluded target in frame 338 (middle) has $OS = 2.46$. Frame 971 (right) has $OS = -0.60$.

tracker's learning rate or not. Given a preexisting learning rate λ_0, the learning rate for the nth frame is

$$\lambda_n = \begin{cases} \lambda_0 & \text{if } OS(n) < \alpha \\ \lambda_0 - \left(\frac{OS(n)-\alpha}{\beta-\alpha}\right)\lambda_0 & \text{if } \alpha < OS(n) < \beta \\ 0 & \text{if } OS(n) > \beta \end{cases} \tag{5.5}$$

where α and β are occlusion score thresholds, and $\beta \geq \alpha$. When $OS(n) < \alpha$, we are declaring that the target is visible and the tracker $\mathcal{T}$should be updated with the target patch from this frame. When $OS(n) > \beta$, we are very confident that the target is occluded and that retraining the tracker $\mathcal{T}$will be detrimental to performance. When $\alpha < OS(n) < \beta$, we are not certain if the target is occluded or not. Setting $\alpha = \beta$ produces a hard threshold and a binary decision to update the tracking model or not. We compare the use of hard

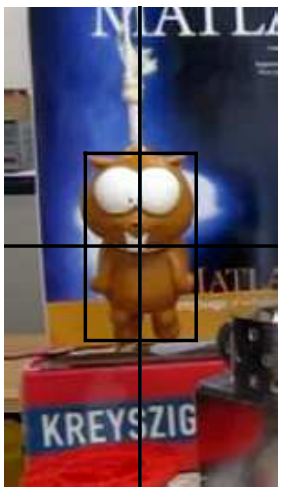


Figure 5.5: Example of how the region of interest is divided into parts for feature weighting. In this initial frame, starting from the top left and moving clockwise, the four interior regions (the target) have raw occlusion scores of -0.87, -0.82, -1.24, and -1.74, while the four exterior regions have raw occlusion scores of 1.76, 2.04, 1.00, and 0.88.

and soft thresholds in Sec. 5.4.

5.3.3 Parts-Based Hue-Based Occlusion Estimation

Computing *occlusion scores* can help reduce the harmful effects of updating a tracker model when the target is occluded, but there are many cases where the target is only partially occluded. In such instances, it may be more helpful to identify separate regions of the detected target that are occluded and unoccluded, rather than treating the target as a whole. If a target is partially exposed, it may be best to update the parts of the model corresponding to visible regions of the target while not updating the occluded portions. Additionally, looking at the hues of smaller regions can allow us apply the HuBOE concepts to the target detection phase of tracking to mask or reduce the weight given to regions that are likely occluded the actual target. This additional step to target detection is not possible if the region of interest is treated as a whole, as in the full-target HuBOE. We refer to the two uses of parts-based HuBOE as *controlling model adaptation* (CA) and *target feature weighting* (TW).

Without specific knowledge of the target being tracked, it is difficult to define a parts model that is customized to that object. Rather than attempt to segment the target region automatically into object-specific parts, we simply divide the target region into a 2×2 grid. For consistency, we also divide the surrounding region into four quadrants, as shown in Fig. 5.5. We do not experiment with further subdivision of the target region or surroundings in this work. While further subdivision and smaller target parts is possible, the possibility of small sample sizes within each part becomes more likely, particularly with low resolution targets that could lead to unstable occlusion estimates. It is also important to note that the log-likelihood for a given hue, as shown in Eq. 5.2, is still computed by considering the target and surrounding regions as a whole. This approach is chosen to allow for in-plane rotations of the target without the increased possibility of being confused as occlusion, in the case that different target parts consist of different hues.

For full-target HuBOE, we use the first frame's occlusion score to normalize subsequent scores (Eq. 5.4). For the part-based approach, it is still important to adjust scores based on the initial target hues while also considering that certain model parts may have less distinctive hues than others. Accordingly, we compute the correcting factor for occlusion scores by accounting for all the target parts and surrounding regions as:

$$OS_{offset} = mean\left(OS_{raw}^{surr}(1)\right) - mean\left(OS_{raw}^{target}(1)\right) \tag{5.6}$$

where $OS_{raw}^{surr}(1)$ and $OS_{raw}^{target}(1)$ denote the raw occlusion scores of the four surrounding regions and four target regions, respectively, in the first frame. Just as with full-target HuBOE, this value is used to normalize scores in subsequent frames as follows:

$$OS^{region}(n) = OS_{raw}^{region}(n) - OS_{offset} \tag{5.7}$$

where $OS_{raw}^{region}(n)$ is the raw occlusion score for an individual region. Similar to full-target HuBOE, this normalized score is used to either let the tracker run as usual or remove the impact of occlusion. Rather than directly reduce the learning rate, we learn a weight $\eta^{region}(n)$ between 0 and 1 for the features within the region:

$$\eta^{region}(n) = \begin{cases} 1 & \text{if } OS^{region}(n) < \gamma \\ 1 - \frac{OS^{region}(n)-\gamma}{\psi-\gamma} & \text{if } \gamma < OS^{region}(n) < \psi \\ 0 & \text{if } OS^{region}(n) > \psi \end{cases} \tag{5.8}$$

where γ and ψ are score thresholds and $\psi \geq \gamma$. The subsequent feature representation is weighted as

$$\hat{f}(I^{region}) = \eta^{region} \cdot f(I^{region}) \tag{5.9}$$

where $f(I^{region})$ is the feature representation for a given region. Thus, a higher $OS^{region}(n)$ will lower the weight $\eta^{region}(n)$ given to regions of the image which likely contain non-target objects and occlusions. As stated earlier, these weights can be computed prior to learning the update for the target model, or they can be learned and applied to a region of interest in a new frame, which can possibly reduce the effect of clutter or occlusions when detecting the target. These two uses are neither dependent nor exclusive of each other, and the different uses are compared in Sec. 5.4.

5.4 Experiments

5.4.1 Experimental Setup

We test HuBOE on color video sequences used in the five OTB and VOT benchmarks. We include all color video sequences, and additionally show results on OTB sequences tagged with occlusion (see Table 5.1 for detailed statistics of each dataset). We augment five publicly available CFTs that follow the basic retraining step outlined in Eq. 5.1. The five trackers are:

Dataset	OTB50	OTB50 (Occ.)	OTB100	OTB100 (Occ.)	VOT 2013	VOT 2014	VOT 2015
Tracks	36	21	77	42	16	25	60
Segments	1342	807	3090	1866	643	957	1722
Frames	18276	13931	44361	28429	6094	10209	20931

Table 5.1: Number of full-color videos, segments, and frames in the benchmark datasets. Videos in the OTB datasets tagged with occlusion are noted separately.

1. *CSK* [3]: The CSK tracker uses pixel intensity features and does not account for a target's scale changes.
2. *KCF* [2]: The KCF tracker uses HOG descriptors and does not account for a target's scale changes.
3. *DSST* [19]: The DSST tracker uses HOG descriptors and does estimate a target's scale changes with the use of a second CF.
4. *SAMF* [23]: The SAMF tracker uses HOG and color name feature descriptors and does estimate a target's scale changes with the use of a second CF.
5. *Staple* [22]: The Staple tracker uses HOG and histograms of RGB color features and does estimate a target's scale changes with the use of a second CF.

In all five cases, the trackers use the provided parameters and are unmodified except for the points where we insert HuBOE into the tracker as shown in Fig. 5.1; first, in changing the adaptation rate λ, as described in Eq. 5.5, and in changing the feature representation of the new frame, as detailed in Eqs. 5.8 and 5.9. The default adaptation rate is unchanged from the original trackers, except when it is reduced to 0 when occlusion is detected. Additionally, we note that the feature representations used for the scale estimators in the DSST, SAMF, and Staple trackers are not modified according to Eqs. 5.8 and 5.9; in all five cases, only the translation estimation is altered. Additionally, when modifying the Staple tracker, which is comprised of a combination of a correlation filter using HOG features and color histograms, we only modify the learning rates of the HOG CF during the course of

Tracker	Frames per Second		
	Baseline	Full HuBOE	Parts HuBOE
CSK [3]	204	183	109
KCF [2]	144	115	92.9
DSST [19]	21.6	21.5	20.8
SAMF [23]	8.93	8.87	8.52
Staple [22]	35.9	35.0	27.2

Table 5.2: Speed of each CFT without HuBOE and with full-target and parts-based HuBOE added.

tracking. With each tracker, we report the baseline performance, the performance when we use full-target HuBOE and the performance of parts-based HuBOE.

Timing results are for trackers run on an Intel(R) Core(TM) i5-4210U CPU using a single core at 1.70 GHz with 8G RAM. The average tracking speeds of the five CFTs with and without HuBOE on the 77 OTB100 tracks can be seen in Table 5.2, and shows that the added computation is minimal.

5.4.2 Results

Quantitative results can be seen in Tables 5.3, 5.4, and 5.5. The values reported in Tables 5.3 and 5.4 are summary statistics showing how often a tracker has a CLE of less than 20 pixels and the AVO, respectively, and Table 5.5 shows the average ASO and reports the statistical significance of introducing HuBOE to the trackers. Plots showing full results on the two largest datasets, OTB100 and VOT2016, can be seen in Figs. 5.6 and 5.7. These results show a consistent improvement to all five trackers with both full-target and parts-based HuBOE. As would be expected, the improvement is most noticeable when considering only videos with occlusion present, as demonstrated on the occlusion subsets of OTB datasets. This trend appears when observing the trackers; examples of each baseline tracker being corrected are shown in Figs. 5.8 and 5.9, and we see that most of the failures that are corrected are at points where the target is occluded. We also note that in most

cases, the quantitative improvement is more pronounced when considering the 20 pixel CLE threshold rather than other measures. This reflects the stated purpose of HuBOE to handle occlusions, which often cause complete tracking failures when not handled. HuBOE is less likely to prevent a tracker from drifting off the target in other conditions.

While HuBOE overall improves all of the trackers, some trends within the results are worth noting. While the five trackers have varying baseline performance levels, the amount of improvement from HuBOE is not limited to just the lowest performing trackers. It would be fair to expect HuBOE gives the biggest gains on the lesser performing trackers simply because there is more room to improve, but this does not seem to be the case in our results. Rather, the KCF tracker generally shows the smallest improvement while the Staple tracker still does not appear to be hitting any limit in performance. It is also notable that SAMF and Staple include color features in their tracker, but still benefit from the addition of HuBOE. These can be easily explained by the separation of the target detection and model updates for each frame. While the SAMF and Staple trackers demonstrate benefits of using color features in their models, they are both still prone to learning the appearance of occluding objects because there is no information from the tracker detection that is fed into a decision to update their model.

Finally, it is worth comparing the results from full-target HuBOE and parts-based HuBOE. Overall, the parts-based approach works better than the full-target approach, but the individual results are dependent on the algorithm and the dataset. In particular, the DSST and SAMF trackers show that the parts-based HuBOE consistently outperforms the full-target HuBOE approach for those algorithms. Additionally, the full-target approach tends to work better on OTB datasets, whereas the parts-based approach works better more often on the VOT datasets. This may indicate that some underlying trends in the data are better handled by one approach or the other, or perhaps that the parts-based approach is more effective on challenging data (such as the VOT2015 dataset), but such

Tracker		OTB50	OTB50 (Occ.)	OTB100	OTB100 (Occ.)	VOT 2013	VOT 2014	VOT 2015
CSK	None	0.516	0.473	0.484	0.395	0.524	0.497	0.345
	Full	0.562	**0.570**	**0.533**	**0.473**	0.577	0.513	0.369
	Parts	**0.583**	0.551	0.513	0.469	**0.603**	**0.534**	**0.382**
KCF	None	0.713	0.703	0.654	0.601	0.710	0.617	0.420
	Full	0.730	0.725	0.675	**0.633**	0.718	0.628	**0.450**
	Parts	**0.740**	**0.744**	**0.676**	0.632	**0.761**	**0.644**	0.440
DSST	None	0.675	0.588	0.625	0.526	0.734	0.622	0.437
	Full	0.718	0.654	0.660	0.586	0.776	0.649	0.477
	Parts	**0.755**	**0.727**	**0.701**	**0.642**	**0.842**	**0.705**	**0.489**
SAMF	None	0.739	0.767	0.714	0.685	0.772	0.665	0.483
	Full	0.795	0.859	0.742	0.730	0.804	0.714	**0.504**
	Parts	**0.797**	**0.866**	**0.747**	**0.750**	**0.808**	**0.732**	0.502
Staple	None	0.762	0.739	0.755	0.704	0.798	0.703	0.511
	Full	0.811	**0.821**	0.795	**0.776**	0.826	0.715	0.544
	Parts	**0.833**	0.808	**0.798**	0.742	**0.830**	**0.723**	**0.558**

Table 5.3: 20 pixel accuracy of 5 CFTs. Includes results with baseline without using HuBOE (None), results using full-target HuBOE (Full), and results using parts-based HuBOE (Parts).

observations are speculative given the uncertainty in these results.

While Tables 5.3 and 5.4 show the overall tracking improvements gained with HuBOE, we also show the performance of specific components of HuBOE discussed in Sec. 5.3.

Comparing ASO and AVO

The ASO metric was introduced in Ch. 3, but generally large discrepancies between the different trackers in Ch. 4 often made the choice of accuracy metric an afterthought; as previously noted, any reasonable metric will produce the same rank-ordering between trackers with a large different in performance. In contrast, trackers with both full-target and the parts-based HuBOE often show much smaller differences in performance. Comparing the results when using AVO in Table 5.4 and AVO in Table 5.5, we can see a number of instances where the rank-ordering of the two HuBOE approaches can change, e.g., the VOT2013 results for both KCF and Staple. In a few instances, the rank-ordering of the

Tracker		OTB50	OTB50 (Occ.)	OTB100	OTB100 (Occ.)	VOT 2013	VOT 2014	VOT 2015
CSK	None	0.367	0.341	0.359	0.303	0.385	0.349	0.254
	Full	0.400	**0.410**	**0.386**	**0.346**	0.423	0.352	0.266
	Parts	**0.404**	0.395	0.376	0.342	**0.426**	**0.376**	**0.286**
KCF	None	0.495	0.493	0.456	0.419	0.506	0.418	0.296
	Full	0.509	**0.515**	**0.477**	**0.451**	0.518	0.438	0.310
	Parts	**0.510**	0.514	0.473	0.441	**0.526**	**0.440**	**0.312**
DSST	None	0.520	0.453	0.484	0.413	0.551	0.489	0.343
	Full	0.541	0.495	0.501	0.447	0.584	0.505	0.374
	Parts	**0.570**	**0.537**	**0.528**	**0.491**	**0.606**	**0.537**	**0.376**
SAMF	None	0.550	0.577	0.517	0.507	0.574	0.493	0.355
	Full	0.581	0.634	0.535	0.537	0.581	0.531	0.362
	Parts	**0.587**	**0.639**	**0.536**	**0.546**	**0.583**	**0.546**	**0.364**
Staple	None	0.588	0.558	0.571	0.537	0.616	0.558	0.402
	Full	**0.629**	**0.625**	0.596	**0.590**	**0.631**	0.556	0.417
	Parts	**0.629**	0.602	**0.602**	0.567	0.628	**0.572**	**0.431**

Table 5.4: AVO of 5 CFTs. Includes results with baseline without using HuBOE (None), results using full-target HuBOE (Full), and results using parts-based HuBOE (Parts).

HuBOE appraoches and the baseline change as well, although this is less frequent and can be attributed the magnitude of the benefits that HuBOE often provides. Beyond complete changes in the rank-ordering, some small differences become much more pronounced when using ASO, e.g., the OTB50 HuBOE results with the CSK tracker. While the motivation for introducing ASO as an alternative to AVO was explained in Ch. 3, these experimental results indicate that the ASO metric has a real impact on how trackers are compared beyond its intuitive meaning.

Along with the rank-ordering differences between AVO and ASO, we also highlight the p-values that are computed to compare each HuBOE tracker with its respective baseline tracker. Some trackers produce the same ASO but different p-values, e.g., the two HuBOE SAMF trackers on the VOT2013 and VOT2015 benchmarks. This provides insight as to how much variance a proposed change may produce, even with the same overall accuracy. The trackers with the smaller p-value is less likely to degrade the performance, relative

Tracker		OTB50	OTB50 (Occ.)	OTB100	OTB100 (Occ.)	VOT 2013	VOT 2014	VOT 2015
CSK	None	0.412	0.416	0.355	0.306	0.441	0.389	0.274
	Full	**0.436**	**0.473**	**0.411**	**0.362**	**0.509**	**0.413**	**0.284**
	p-value	< 0.001	< 0.001	< 0.001	< 0.001	< 0.001	< 0.001	< 0.001
	Parts	**0.473**	**0.510**	**0.429**	**0.383**	**0.503**	**0.421**	**0.336**
	p-value	< 0.001	< 0.001	< 0.001	< 0.001	< 0.001	< 0.001	< 0.001
KCF	None	0.549	0.539	0.451	0.410	0.596	0.492	0.340
	Full	0.554	0.536	**0.484**	**0.462**	0.600	0.506	**0.381**
	p-value	> 0.05	–	< 0.001	< 0.001	> 0.05	0.003	< 0.001
	Parts	**0.555**	**0.549**	**0.477**	**0.446**	**0.606**	**0.506**	**0.366**
	p-value	< 0.001	< 0.001	< 0.001	< 0.001	< 0.001	< 0.001	< 0.001
DSST	None	0.572	0.476	0.490	0.399	0.641	0.577	0.383
	Full	**0.588**	**0.501**	**0.556**	**0.476**	**0.647**	0.576	0.384
	p-value	< 0.001	< 0.001	< 0.001	< 0.001	< 0.001	–	> 0.05
	Parts	**0.607**	**0.546**	**0.564**	**0.490**	**0.670**	**0.622**	**0.450**
	p-value	< 0.001	< 0.001	< 0.001	< 0.001	< 0.001	< 0.001	< 0.001
SAMF	None	0.603	0.626	0.539	0.530	0.669	0.592	0.401
	Full	**0.628**	**0.668**	**0.549**	**0.549**	0.675	**0.612**	0.413
	p-value	< 0.001	< 0.001	< 0.001	< 0.001	0.027	< 0.001	0.016
	Parts	**0.630**	**0.671**	**0.566**	**0.551**	**0.675**	**0.633**	**0.413**
	p-value	< 0.001	< 0.001	< 0.001	< 0.001	< 0.001	< 0.001	< 0.001
Staple	None	0.624	0.578	0.573	0.505	0.704	0.663	0.425
	Full	**0.672**	**0.654**	**0.608**	**0.598**	**0.716**	0.668	**0.468**
	p-value	< 0.001	< 0.001	< 0.001	< 0.001	< 0.001	> 0.05	< 0.001
	Parts	**0.676**	**0.615**	**0.621**	**0.557**	**0.719**	0.659	**0.512**
	p-value	< 0.001	< 0.001	< 0.001	< 0.001	< 0.001	–	< 0.001

Table 5.5: Average ASO of 5 CFTs. Includes results with baseline without using HuBOE (None), results using full-target HuBOE (Full), and results using parts-based HuBOE (Parts), and computed p-values for paired t-tests when adding HuBOE with respect to the baseline trackers.

to the baseline, on a given segment than the tracker with the higher p-value, which likely improved the performance on more segments but also reduced the accuracy on more other segments. As discussed in Ch. 3, the statistical testing to produce the p-values depends on the independence of each trial. While segments within a single video are not truly independent, they approach independence much more so than when each individual frame is considered a trial with its own overlap value. In this manner, the ASO provides another insight and another benefit over the use of AVO.

Selecting Occlusion Score Thresholds

Figs. 5.10 and 5.11 show results for full-target HuBOE when using a range of different values for α and β on OTB100 and VOT2015 datasets. We tested ranges $0 \leq \alpha \leq 1$ and $\alpha \leq \beta \leq \alpha + 1$. In our observation, setting $\alpha = 1$ is a conservative choice, and such a high threshold will result in HuBOE changing the adaptation rate very infrequently; for this reason, we limit our testing to $\alpha \leq 1$.

The results in Tables 5.3 and 5.4 highlight the best results when using HuBOE, while Figs. 5.10 and 5.11 show the overall sensitivity of these parameters. In most cases, the choice of α and β is not very sensitive, but there are some irregular peaks and valleys with different trackers. Such an outcome should be expected to some degree with any tracking parameter, when small changes in the model can create large changes in performance metrics if such a change is the difference between failing early in a video or not.

Beyond the overall sensitivity of α and β, we observe a trend that lower thresholds, i.e., more aggressive use of the HuBOE system does in fact harm the performance in most cases. This is not surprising; as $\alpha \to 0$, we are suggesting that any occlusion score that is higher than the occlusion score from the first frame, i.e., the unoccluded ground truth frame, indicates some amount of occlusion. This is a very aggressive interpretation of the occlusion scores, and accordingly the performance is harmed by it.

Tracker		OTB50	OTB50 (Occ.)	OTB100	OTB100 (Occ.)	VOT 2013	VOT 2014	VOT 2015
CSK	Hard	0.532	0.505	0.522	0.449	0.534	0.494	0.355
	Soft	**0.562**	**0.570**	**0.533**	**0.473**	**0.577**	**0.513**	**0.369**
KCF	Hard	0.727	0.705	0.664	0.609	0.709	0.621	0.443
	Soft	**0.730**	**0.725**	**0.675**	**0.633**	**0.718**	**0.628**	**0.450**
DSST	Hard	0.715	0.649	0.657	0.581	**0.776**	0.644	0.466
	Soft	**0.718**	**0.654**	**0.660**	**0.586**	**0.776**	**0.649**	**0.477**
SAMF	Hard	0.772	0.824	0.728	0.720	0.787	0.689	0.503
	Soft	**0.795**	**0.859**	**0.742**	**0.730**	**0.804**	**0.714**	**0.504**
Staple	Hard	0.797	0.802	0.782	0.750	0.821	**0.715**	0.531
	Soft	**0.811**	**0.821**	**0.795**	**0.776**	**0.826**	**0.715**	**0.544**

Table 5.6: Comparison of 20 pixel accuracies when using hard and soft thresholds for full-target HuBOE.

Hard vs. Soft Threshold

Table 5.6 shows results with full-target HuBOE approach outlined by Eq. 5.5 using both a soft threshold for reducing the learning rate of a CFT and a hard threshold ($\alpha = \beta$). While the full-target HuBOE outlined in Eq. 5.5 does not require that $\beta > \alpha$, the results show that it is beneficial across all trackers and all datasets to have a soft threshold that allows for uncertainty when the occlusion score is within a certain range. Though not shown, similar benefits when using a soft threshold hold when using the parts-based HuBOE.

Parts-Based Approaches

In Sec. 5.3, we discussed two ways to apply HuBOE to a parts-based approach: by weighting the features that are fed into the model adaptation (denoted as CA, *controlling adaptation*) and by weighting the features extracted from the new frame where the model will detect the target (denoted as TW, *target weighting*). Table 5.7 shows the results of each approach, as well as using both together. Overall, using the HuBOE only for weighting the model adaptation works best for the most cases. When using both applications of parts-based HuBOE together, the same score thresholds ψ and γ are used for both the CA and TW

components, though that may not be ideal, which can explain why using both together sometimes performs worse than both the CA and TW components individually. Although not shown side-by-side, it is important to note that nearly all approaches are better than the baseline trackers, with the exception of some KCF results and the Staple+TW approach on VOT2014. The CA parts-based HuBOE approach outperforms the baseline tracker in all cases.

An explanation why the CA application is more effective compared to the TW application of HuBOE can be tied back to the strengths and weaknesses of the baseline trackers. The tracking models are designed to distinguish between target features and non-target features, whether they are pixel intensities (CSK), texture features (all), or color features (SAMF, Staple). The TW application reduces the weight of features in regions with non-target hues, but these hues are already explicitly identified as non-target by the SAMF and Staple trackers, and the texture features of occluding objects are, if not guaranteed, at least likely to be different from the original target. The weakness of these trackers is not that they confused occluding objects with the original target, it is that the trackers are forced to learn the appearance of the occluding objects if the target is obscured for too long. This weakness is more directly addressed by the CA application of parts-based HuBOE (and also by full-target HuBOE) and therefore the CA application should be expected to perform better. At the same time, the TW application may still provide a lesser benefit and not actually harm the tracking performance, which is what is observed.

5.5 Chapter Summary

We introduced HuBOE, which makes inferences about portions of video sequences that likely belong to occlusion in visual tracking tasks. It can be added to existing trackers, including but not limited to the CFTs demonstrated in this work, that have a retraining model that uses a simple linear combination of a current tracking model and an update

Tracker		OTB50	OTB50 (Occ.)	OTB100	OTB100 (Occ.)	VOT 2013	VOT 2014	VOT 2015
CSK	CA	**0.583**	0.546	**0.513**	0.460	**0.603**	0.521	0.378
	TW	0.537	0.535	0.496	0.446	0.587	**0.534**	**0.382**
	Both	0.554	**0.551**	0.503	**0.469**	0.584	0.526	0.376
KCF	CA	**0.740**	**0.744**	0.669	0.627	**0.761**	**0.644**	0.425
	TW	0.732	0.725	**0.676**	**0.632**	0.755	0.622	**0.440**
	Both	0.717	0.688	0.653	0.598	0.722	0.615	0.403
DSST	CA	**0.755**	0.715	**0.701**	0.630	**0.842**	**0.705**	**0.489**
	TW	0.701	0.644	0.650	0.566	0.752	0.650	0.462
	Both	0.749	**0.727**	0.694	**0.642**	0.818	0.695	0.488
SAMF	CA	0.795	**0.866**	**0.747**	**0.750**	**0.808**	**0.732**	0.494
	TW	**0.797**	0.865	0.741	0.736	0.797	0.729	**0.502**
	Both	0.782	0.862	0.734	0.736	0.798	0.730	0.493
Staple	CA	0.814	0.768	**0.798**	**0.742**	**0.830**	**0.723**	**0.558**
	TW	0.784	0.777	0.761	0.727	0.811	0.702	0.536
	Both	**0.833**	**0.808**	0.785	0.730	0.828	**0.723**	0.541

Table 5.7: 20 pixel accuracy of 5 CFTs using parts-based HuBOE. Includes results when using the parts-based feature weighting to control the model adaptation (CA), when performing parts-based target weighting (TW), and both.

built on a new detection.

Using five CFTs, we showed the effectiveness of HuBOE on the popular benchmark tracking datasets. HuBOE is able to identify occlusions and prevent critical tracking errors from occurring without adding much computational time. Although the tracking datasets show the broad utility of HuBOE, specific applications could modify the framework presented in this work. In particular, the parts-based approach could be tailored to the specific structure of a known target. Additionally, one change to trackers not explored is to increase the base learning rate. If frames containing occlusion can be avoided in the model adaptation phase of tracking, then the learning rates can possibly be more aggressive to keep up with fast changes in a target's appearance.

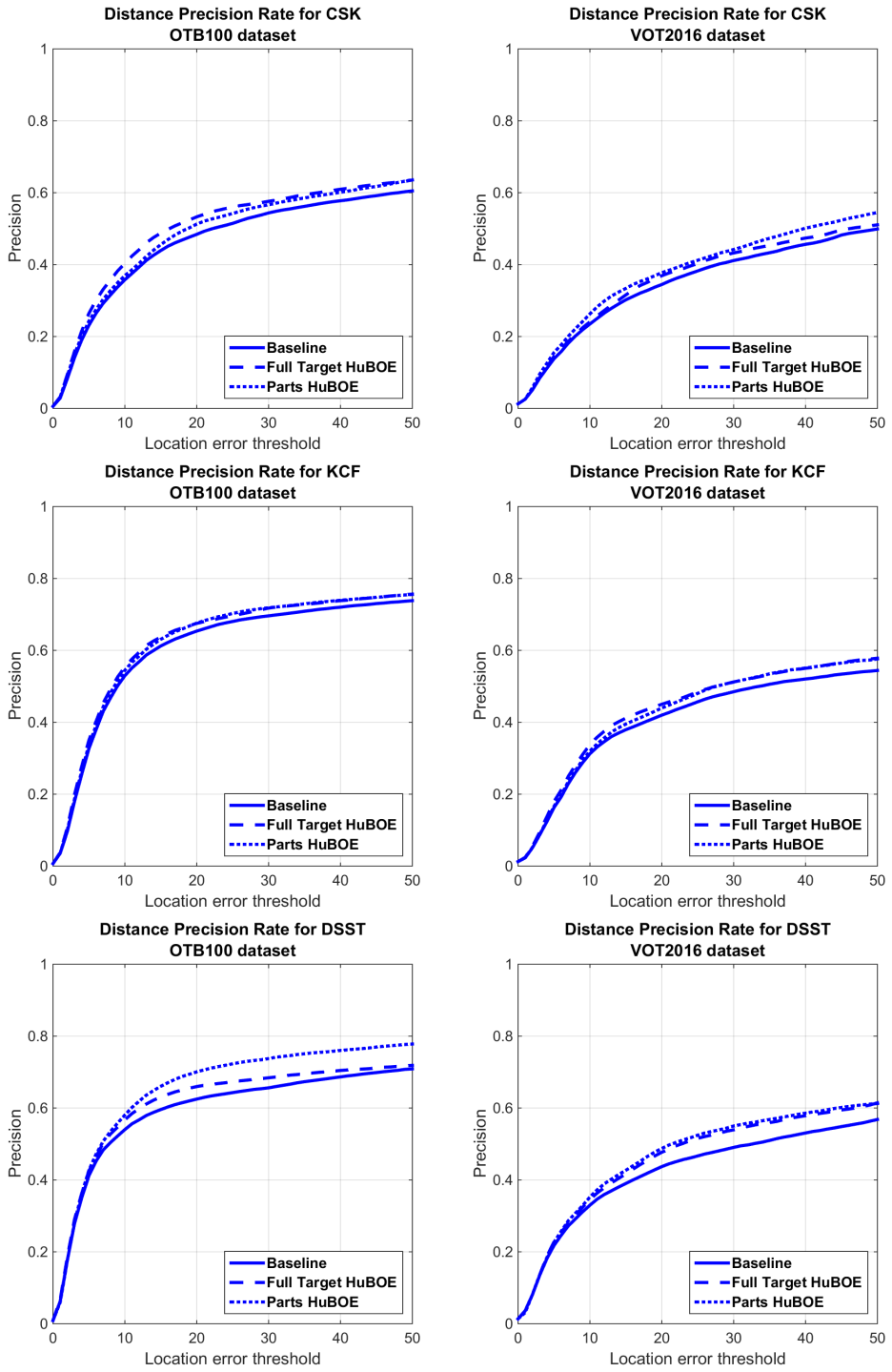


Figure 5.6: Results for CSK, KCF, and DSST trackers on OTB100 and VOT2016 datasets. "Baseline" refers to the original tracker with no HuBOE used, "full target HuBOE" refers to using HuBOE to the entire target region, and "parts HuBOE" refers to using the parts based approach to occlusion detection. Plots for each dataset show the average distance precision.

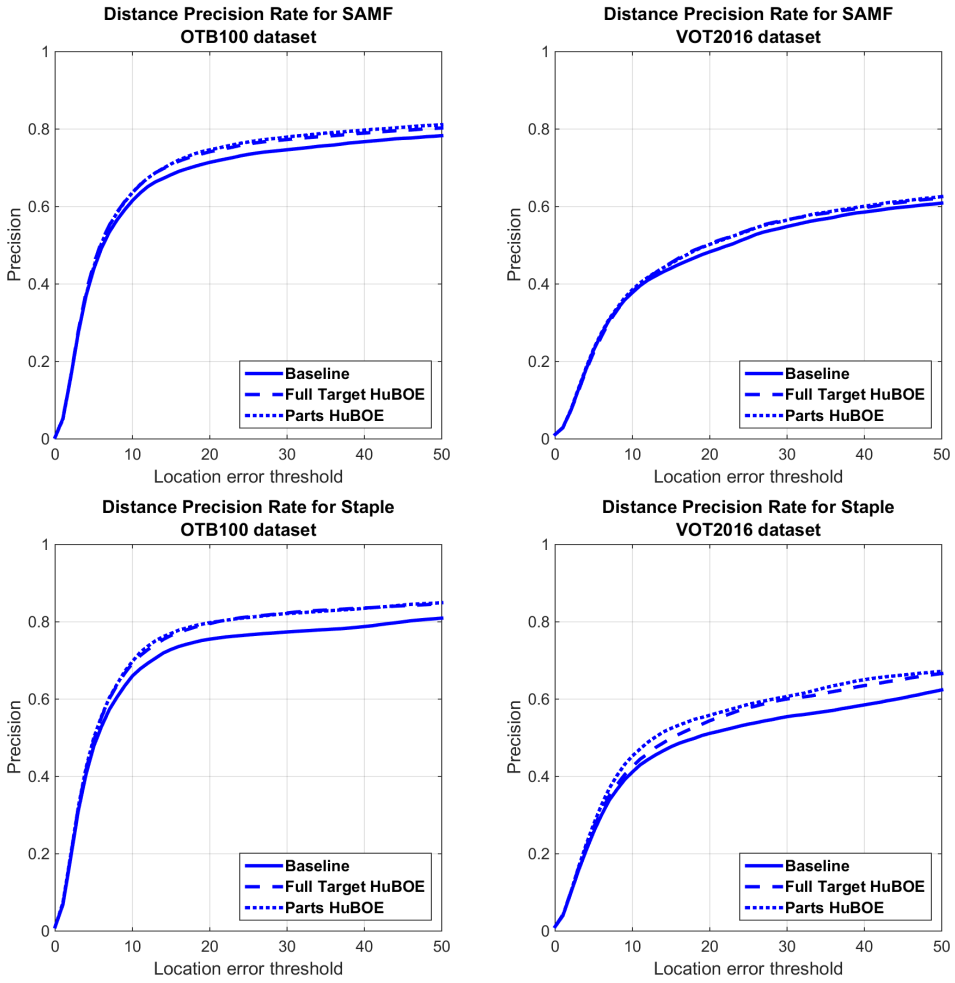
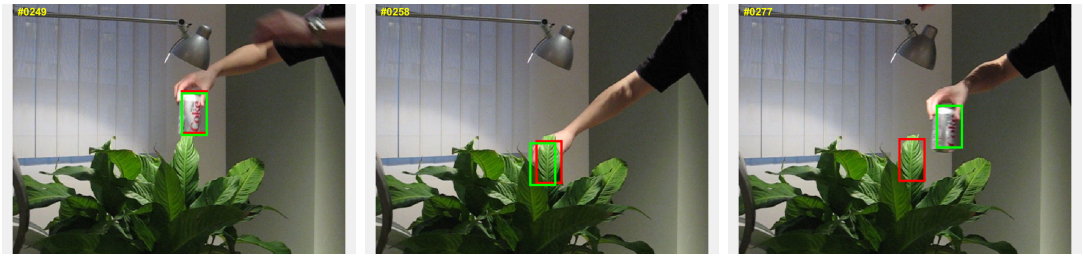


Figure 5.7: Results for SAMF, and Staple trackers on OTB100 and VOT2016 datasets. "Baseline" refers to the original tracker with no HuBOE used, "full target HuBOE" refers to using HuBOE to the entire target region, and "parts HuBOE" refers to using the parts based approach to occlusion detection. Plots for each dataset show the average distance precision.

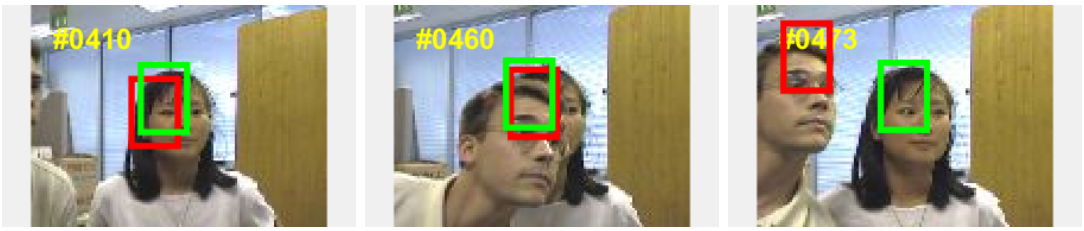
(a) ‘road’ sequence, CSK tracker

(b) ‘woman’ sequence, CSK tracker



(c) ‘coke’ sequence, KCF tracker

(d) ‘lemming’ sequence, KCF tracker



(e) ‘girl’ sequence, DSST tracker

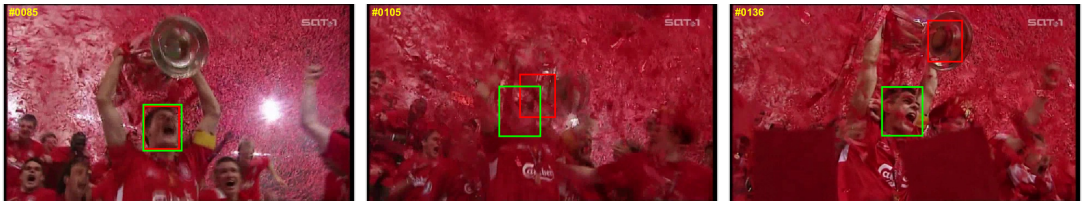
Figure 5.8: Examples of CSK, KCF, and DSST trackers failing due to occlusion and being corrected with HuBOE. Each image sequence shows the target immediately before being occluded, during occlusion, and shortly after reappearing. The red box denotes the original tracker and green box denotes that HuBOE is included.

(a) ‘subway’ sequence, DSST tracker

(b) ‘jogging’ sequence, SAMF tracker

(c) ‘tiger1’ sequence, SAMF tracker

(d) ‘skiing’ sequence, Staple tracker



(e) ‘soccer’ sequence, Staple tracker

Figure 5.9: Examples of DSST, SAMF, and Staple trackers failing due to occlusion and being corrected with HuBOE. Each image sequence shows the target immediately before being occluded, during occlusion, and shortly after reappearing. The red box denotes the original tracker and green box denotes that HuBOE is included.

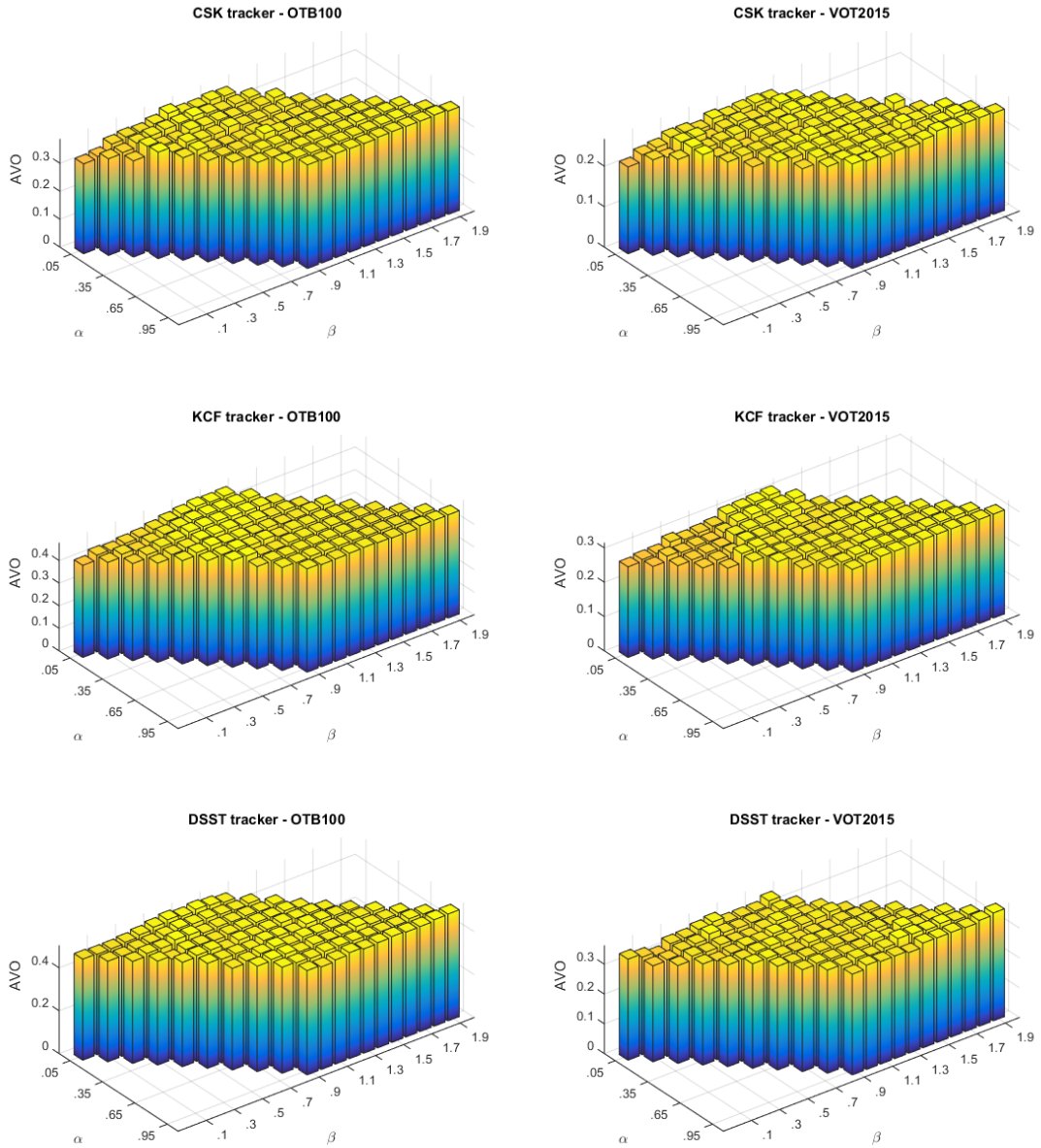


Figure 5.10: AVO on OTB100 and VOT2015 for CSK, KCF, and DSST trackers when adjusting the occlusion score thresholds α and β for full-target HuBOE. Results in the left-hand corner of each plot represent more aggressive thresholds that will result in more changes in the adaptation rate, while results in the right-hand side of each plot represent less aggressive thresholds that will trigger less changes in the adaptation rate for a given tracker.

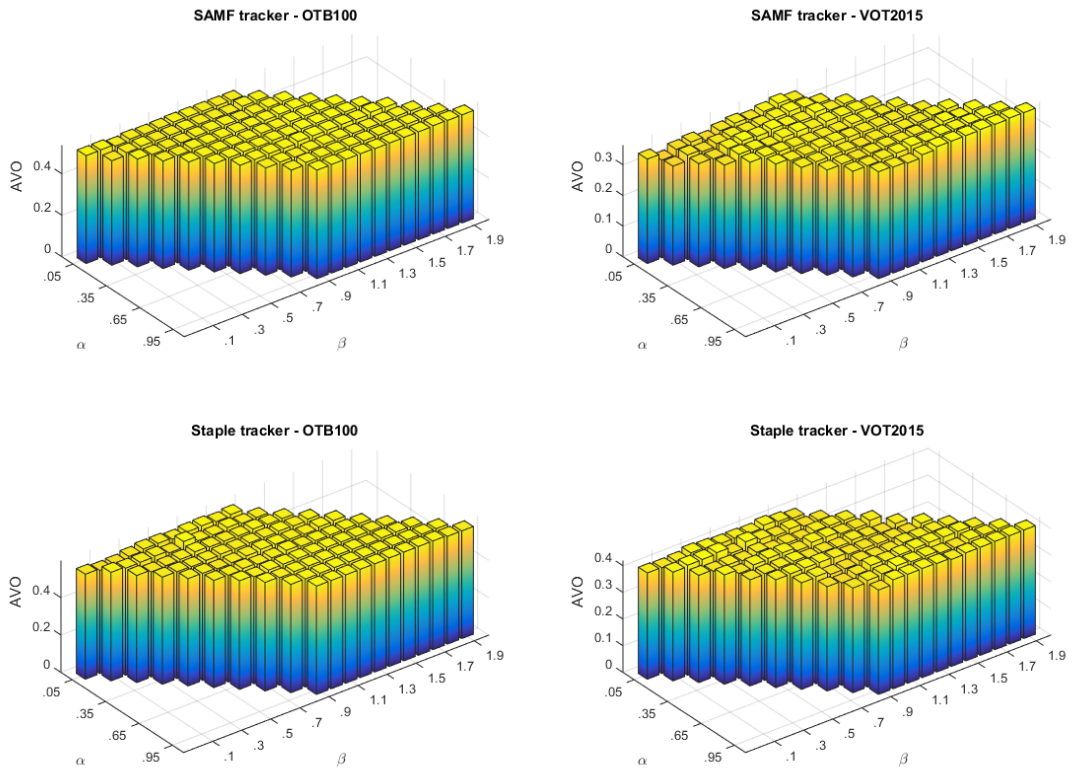


Figure 5.11: AVO on OTB100 and VOT2015 for SAMF and Staple trackers when adjusting the occlusion score thresholds α and β for full-target HuBOE.

Chapter 6

Conclusion

In this chapter, we review the work presented in the previous chapters, and highlight the main contributions of the work. We finish with a discussion of future directions of research.

6.1 Thesis Summary

Chapter 1 introduced the problem of model-free visual tracking, and Chapter 2 discussed advances in CFTs, which have become one of the prevailing approaches in visual tracking. Chapter 2 also discussed the progression from very fundamental CFs to designs that have been effective for a range of localization tasks, which have been only partially adapted to visual tracking.

Chapter 3 discussed some of the most common benchmarks used to evaluate visual trackers. Benchmark datasets have contributed greatly to the shared knowledge of new visual trackers; large aggregations of tracking videos with standardized performance measures have made comparisons between different algorithms easier while mitigating dataset selection biases. However, we feel that the performance measures – based on per frame accuracies – do not exactly capture and measure the challenge trackers face. Rather than treat each successive frame as a new trial to measure trackers on, we proposed ASO, which aggregates

the frame overlap within homogenous temporal segments of the video. ASO follows the intuition that the tracker performance is determined when the targets undergo some change that would be expected to challenge the tracker, e.g., an occlusion or a lighting change. Each single value of ASO starts and stops at a point human observers naturally watch for with the most anticipation – the next test in the video that the tracker must pass. We also proposed a way to automatically segment videos to be able to compute the ASO without human annotation.

Chapter 4 explored the use of CF designs not previously explored at any depth for visual tracking. While the MOSSE filter and its kernelized version, KCF, have been used in a number of trackers, other powerful filters designs, including the OTSDF, UOTSDF, and MMCF filters, had apparently not been adapted to visual tracking. We showed that while these filters do not perform well on their own in a tracking system, they can complement the more commonly used KCF filter, particularly on the videos that are most challenging for the KCF tracker. While overall performance measures were not tremendously improved, the specific use case suggests that the fusion of multiple CF designs may have increased effectiveness as tracking data becomes more challenging, either in continually more difficult benchmarks or in challenging real-world conditions. We also discussed the interactions between chosen feature descriptors and filter design.

Chapter 5 addressed the problem of occlusion in visual tracking; one of the most challenging elements in a setting where a vision model is attempting to both continuously locate a target and learn its appearance from almost no training data. We introduce HuBOE, a color-based system for inferring occlusion that is both lightweight and portable to a number of trackers. By only operating on the tracker's bounding box output and by only preventing (or not preventing) a tracker from updating its detection model, HuBOE is versatile enough to improve a number of trackers. We validate HuBOE by incorporating it into a variety of CFTs, including CFTs that already use color features in their target

detection, and show improvements when added to each tracker. We propose two versions of HuBOE that are modeled to find either full or partial occlusion of targets.

6.2 Contributions

The main contributions of the thesis are as follows:

- Propose a temporal track segmentation method and propose ASO, which closely aligns with humans' qualitative evaluation of trackers and captures the amount of activity that occurs within a video.
- Explored the application of CF designs (OTSDF, UOTSDF, MMCF) that had not been used prior in visual tracking. In particular, we found that these trackers worked well as an addition to the existing KCF tracker for the tracks it struggles with most. The UOTSDF filter showed the most promise in this regard.
- Illustrated the overlapping and perhaps redundant effect that more powerful CF designs and more powerful feature representations have on visual trackers.
- Introduced HuBOE, a lightweight occlusion estimator that helps model-free trackers avoid bad model updates when targets are obscured.
- Introduced a parts-based HuBOE for managing partial occlusion appropriately during tracking.

6.3 Future Work

Visual tracking and CFTs are rapidly changing fields, and there are many possible directions for future research:

- As the impact and prevalence of deep learning and CNNs grows in model-free visual tracking, it is increasingly necessary to understand the role and importance CFs may

or may not have within them. Already a number of deep network CFTs exist and perform very well. While the MOSSE filter design is usually used to produce the final outputs of these trackers, it may be still worth exploring if it is the optimal choice. As trackers become more and more sophisticated, there is less and less emphasis on the speed of the tracker. In deep network CFTs, a different CF design may not be a bottleneck for speed.

- Learning-based methods in computer vision broadly have three components: the data for training (and testing), the features chosen to represent images, and the classifier that is learned and ultimately makes decisions. In Chapter 4, we note that the MOSSE filter (i.e., the classifier) performs much more poorly than other CF designs with raw pixel intensity features, but has roughly equal performance when HOG features are used to track objects. This is contradictory to previous results, but previous results were in other applications such as ATR, and had different, larger, training sets. While prior work with different data and features was a motivation for adapting certain filters to visual tracking, the mixed results on visual tracking with more powerful multi-channel features may indicate that it is worth revisiting prior assumptions about filter design and selection in those original problem settings. As multi-channel CFs and richer feature descriptors including deep networks become more and more prevalent, it may be that the recommendations for CF designs will have shifted from prior research.
- HuBOE was shown to be effective on a range of different CFTs that use a linear interpolation update scheme. Other trackers, including some deep network CFTs, store a bank of prior samples to aid in model updates; the effectiveness of HuBOE to prune this set could determine its applicability and effectiveness on some of the best performing trackers.
- As was discussed in Chapter 3, just as there is no single correct way to segment an

image, there is no single correct way to temporally segment a video. We proposed a segmentation that was binary, as it fit with our needs in computing the ASO of a video. While it is a step in the right direction, there may be ways to characterize subtler changes in a target's track in a way that is both more significant than a completely homogeneous video segment, but less than the most dramatic changes seen throughout some tracking videos.

Appendix A

Optimizing a Quadratic Subject to Linear Constraints

Our goal is to minimize a quadratic with linear constraints, which takes the following form:

$$\begin{aligned} \min_{\hat{\mathbf{h}}} \quad & \hat{\mathbf{h}}^{\dagger}\hat{\mathbf{T}}\hat{\mathbf{h}} \\ s.t. \quad & \hat{\mathbf{X}}^{\dagger}\hat{\mathbf{h}} = \mathbf{u}, \end{aligned} \tag{A.1}$$

where $\hat{\mathbf{T}}$ is assumed to be Hermitian. Using Lagrangian multipliers, we can combine the quadratic term with the constraints as a single equation, i.e.,

$$\mathcal{L}(\mathbf{h}, \Lambda) = \hat{\mathbf{h}}^{\dagger}\hat{\mathbf{T}}\hat{\mathbf{h}} - 2\Lambda^{\dagger}(\hat{\mathbf{X}}^{\dagger}\hat{\mathbf{h}} - \mathbf{u}) \tag{A.2}$$

where Λ are the Lagrangian multipliers. To find the minimum with respect to $\hat{\mathbf{h}}$, we take the gradient of $\mathcal{L}(\mathbf{h}, \Lambda)$ with respect to $\hat{\mathbf{h}}$ and set it to $\mathbf{0}$, i.e.,

$$\frac{d\mathcal{L}(\mathbf{h}, \Lambda)}{d\hat{\mathbf{h}}} = 2\hat{\mathbf{T}}\hat{\mathbf{h}} - 2\hat{\mathbf{X}}\Lambda = \mathbf{0}, \tag{A.3}$$

and solving for $\hat{\mathbf{h}}$ gives

$$\hat{\mathbf{h}} = \hat{\mathbf{T}}^{-1}\hat{\mathbf{X}}\Lambda, \tag{A.4}$$

at which point we must determine Λ. Substituting $\hat{\mathbf{h}}$ into the linear constraints from Eq. A.1 gives

$$\begin{aligned}\hat{\mathbf{X}}^{\dagger}\hat{\mathbf{h}} &= \mathbf{u} \\ \hat{\mathbf{X}}^{\dagger}(\hat{\mathbf{T}}^{-1}\hat{\mathbf{X}}\Lambda) &= \mathbf{u}\end{aligned} \tag{A.5}$$

and solving for Λ gives

$$\Lambda = (\hat{\mathbf{X}}^{\dagger}\hat{\mathbf{T}}^{-1}\hat{\mathbf{X}})^{-1}\mathbf{u}. \tag{A.6}$$

Finally, we substitute from Eq. A.6 into Eq. A.4 which gives the solution,

$$\begin{aligned}\hat{\mathbf{h}} &= \hat{\mathbf{T}}^{-1}\hat{\mathbf{X}}\Lambda \\ &= \hat{\mathbf{T}}^{-1}\hat{\mathbf{X}}(\hat{\mathbf{X}}^{\dagger}\hat{\mathbf{T}}^{-1}\hat{\mathbf{X}})^{-1}\mathbf{u}.\end{aligned} \tag{A.7}$$

Bibliography

[1] D. S. Bolme, J. R. Beveridge, B. Draper, Y. M. Lui *et al.*, "Visual object tracking using adaptive correlation filters," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, 2010, pp. 2544–2550.

[2] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista, "High-speed tracking with kernelized correlation filters," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 2015.

[3] ——, "Exploiting the circulant structure of tracking-by-detection with kernels," in *proceedings of the European Conference on Computer Vision*, 2012.

[4] R. Kerekes and B. V. K. Vijaya Kumar, "Enhanced video-based target detection using multi-frame correlation filtering," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 45, no. 1, pp. 289–307, 2009.

[5] B. V. K. Vijaya Kumar, A. Mahalanobis, and R. D. Juday, *Correlation pattern recognition*. Cambridge University Press, 2005.

[6] C. F. Hester and D. Casasent, "Multivariant technique for multiclass pattern recognition," *Applied Optics*, vol. 19, no. 11, pp. 1758–1761, Jun 1980.

[7] B. V. K. Vijaya Kumar, "Minimum-variance synthetic discriminant functions," *Journal of the Optical Society of America A*, vol. 3, no. 10, pp. 1579–1584, Oct 1986.

[8] A. V. Oppenheim, R. W. Schafer, and J. R. Buck, *Discrete-time signal processing*. Prentice Hall, 2009.

[9] A. Mahalanobis, B. V. K. Vijaya Kumar, and D. Casasent, "Minimum average correlation energy filters," *Applied Optics*, vol. 26, no. 17, pp. 3633–3640, Sep 1987.

[10] P. Refregier, "Filter design for optical pattern recognition: multicriteria optimization approach," *Optics Letters*, vol. 15, no. 15, pp. 854–856, Aug 1990.

[11] P. Réfrégier and J. Figue, "Optimal trade-off filters for pattern recognition and their comparison with the wiener approach." *Optical Computing & Processing*, vol. 1, no. 3, pp. 245–266, 1991.

[12] J. Figue and P. Réfrégier, "Optimality of trade-off filters," *Appl. Opt.*, vol. 32, no. 11, pp. 1933–1935, Apr 1993.

[13] A. Rodriguez, V. N. Boddeti, B. V. K. Vijaya Kumar, and A. Mahalanobis, "Maximum margin correlation filter: A new approach for localization and classification," *Image Processing, IEEE Transactions on*, vol. 22, no. 2, pp. 631–643, 2013.

[14] R. Rifkin, G. Yeo, and T. Poggio, "Regularized least-squares classification," *Nato Science Series Sub Series III Computer and Systems Sciences*, vol. 190, pp. 131–154, 2003.

[15] V. Boddeti, T. Kanade, and B. V. K. Vijaya Kumar, "Correlation filters for object alignment," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 2291–2298.

[16] H. Galoogahi, T. Sim, and S. Lucey, "Multi-channel correlation filters," in *Proceedings of the IEEE International Conference on Computer Vision*, 2013, pp. 3072–3079.

[17] A. Rodriguez and B. V. K. Vijaya Kumar, "Dealing with circular correlation effects," in *Proc. SPIE Conf. Autom. Target Recogn.*, vol. 8744, 2013, pp. 87 440N–87 440N.

[18] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1, June 2005, pp. 886–893 vol. 1.

[19] M. Danelljan, G. Häger, F. Khan, and M. Felsberg, "Accurate scale estimation for robust visual tracking," in *British Machine Vision Conference, Nottingham, September 1-5, 2014*, 2014.

[20] C. Ma, X. Yang, C. Zhang, and M.-H. Yang, "Long-term correlation tracking," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[21] O. Akin, E. Erdem, A. Erdem, and K. Mikolajczyk, "Deformable part-based tracking by coupled global and local correlation filters," *Journal of Visual Communication and Image Representation*, vol. 38, pp. 763 – 774, 2016.

[22] L. Bertinetto *et al.*, "STAPLE: Complementary learners for real-time tracking," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[23] Y. Li and J. Zhu, "A scale adaptive kernel correlation filter tracker with feature integration," in *Computer Vision-ECCV 2014 Workshops*. Springer, 2014, pp. 254–265.

[24] G. Zhu, J. Wang, Y. Wu, and H. Lu, "Collaborative correlation tracking," in *Proceedings of the British Machine Vision Conference (BMVC)*, September 2015, pp. 184.1–184.12.

[25] Z. Hong, Z. Chen, C. Wang, X. Mei, D. Prokhorov, and D. Tao, "Multi-store tracker (MUSTer): A cognitive psychology inspired approach to object tracking," in *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*, June 2015, pp. 749–758.

[26] M. Danelljan, G. Hger, F. S. Khan, and M. Felsberg, "Learning spatially regularized correlation filters for visual tracking," in *2015 IEEE International Conference on Computer Vision (ICCV)*, Dec 2015, pp. 4310–4318.

[27] T. Liu, G. Wang, and Q. Yang, "Real-time part-based visual tracking via adaptive correlation filters," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015, pp. 4902–4912.

[28] M. Tang and J. Feng, "Multi-kernel correlation filter for visual tracking," in *2015 IEEE International Conference on Computer Vision (ICCV)*, Dec 2015, pp. 3038–3046.

[29] X. Li, Q. Liu, Z. He, H. Wang, C. Zhang, and W.-S. Chen, "A multi-view model for visual tracking via correlation filters," *Knowledge-Based Systems*, vol. 113, pp. 88 – 99, 2016.

[30] A. S. Montero, J. Lang, and R. Laganire, "Scalable kernel correlation filter with sparse feature integration," in *2015 IEEE International Conference on Computer Vision Workshop (ICCVW)*, Dec 2015, pp. 587–594.

[31] M. Kristan *et al.*, "The visual object tracking VOT2014 challenge results," in *ECCV 2014 Workshops*, 2014, pp. 191–217.

[32] M. Danelljan, F. S. Khan, M. Felsberg, and J. van de Weijer, "Adaptive color attributes for real-time visual tracking," in *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, 2014, pp. 1090–1097.

[33] B. Berlin and P. Kay, *Basic color terms: Their universality and evolution.* Univ of California Press, 1991.

[34] J. Van De Weijer, C. Schmid, J. Verbeek, and D. Larlus, "Learning color names for real-world applications," *IEEE Transactions on Image Processing*, vol. 18, no. 7, pp. 1512–1523, 2009.

[35] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[36] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[37] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[38] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman, "Return of the devil in the details: Delving deep into convolutional nets," in *British Machine Vision Conference*, 2014.

[39] M. Danelljan, G. Hager, F. Shahbaz Khan, and M. Felsberg, "Convolutional features for correlation filter based visual tracking," in *Proceedings of the IEEE International Conference on Computer Vision Workshops*, 2015, pp. 58–66.

[40] C. Ma, J.-B. Huang, X. Yang, and M.-H. Yang, "Hierarchical convolutional features for visual tracking," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 3074–3082.

[41] M. Danelljan, A. Robinson, F. Shahbaz Khan, and M. Felsberg, "Beyond correlation filters: Learning continuous convolution operators for visual tracking," in *14th European Conference on Computer Vision*, 2016.

[42] J. Johnander, M. Danelljan, F. S. Khan, and M. Felsberg, "Dcco: Towards deformable continuous convolution operators for visual tracking," in *Computer Analysis of Images and Patterns: 17th International Conference, CAIP 2017, Ystad, Sweden, August*

22-24, 2017, Proceedings, Part I. Springer International Publishing, 2017.

[43] L. Wang, W. Ouyang, X. Wang, and H. Lu, "Visual tracking with fully convolutional networks," in *2015 IEEE International Conference on Computer Vision (ICCV)*, Dec 2015, pp. 3119–3127.

[44] ——, "STCT: Sequentially training convolutional networks for visual tracking," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 1373–1381.

[45] M. Kristan *et al.*, "The visual object tracking VOT2016 challenge results," in *ECCV 2016 Workshops*, 2016, pp. 777–823.

[46] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part-based models," *IEEE transactions on pattern analysis and machine intelligence*, vol. 32, no. 9, pp. 1627–1645, 2010.

[47] M. Zhang, J. Xing, J. Gao, X. Shi, Q. Wang, and W. Hu, "Joint scale-spatial correlation tracking with adaptive rotation estimation," in *2015 IEEE International Conference on Computer Vision Workshop (ICCVW)*, Dec 2015, pp. 595–603.

[48] T. F. Cootes, G. J. Edwards, and C. J. Taylor, "Active appearance models," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 23, no. 6, pp. 681–685, 2001.

[49] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part-based models," *IEEE transactions on pattern analysis and machine intelligence*, vol. 32, no. 9, pp. 1627–1645, 2010.

[50] M. D. Breitenstein, F. Reichlin, B. Leibe, E. Koller-Meier, and L. Van Gool, "Robust tracking-by-detection using a detector confidence particle filter," in *Computer Vision, 2009 IEEE 12th International Conference on*. IEEE, 2009, pp. 1515–1522.

[51] Z. Kalal, K. Mikolajczyk, and J. Matas, "Tracking-learning-detection," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 34, no. 7, 2012.

[52] ——, "Forward-backward error: Automatic detection of tracking failures," in *Pattern recognition (ICPR), 2010 20th international conference on*. IEEE, 2010, pp. 2756–2759.

[53] Y. Wu, J. Lim, and M.-H. Yang, "Online object tracking: A benchmark," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013.

[54] M. Rapuru, S. Kakanuru, P. Venugopal, D. Mishra, and G. R. Subrahmanyam, "Correlation based tracker level fusion for robust visual tracking," *IEEE Transactions on Image Processing*, 2017.

[55] W. Zhong, H. Lu, and M.-H. Yang, "Robust object tracking via sparsity-based collaborative model," in *Computer vision and pattern recognition (CVPR), 2012 IEEE Conference on*. IEEE, 2012, pp. 1838–1845.

[56] 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*, vol. 38, no. 10, pp. 2096–2109, Oct 2016.

[57] H. Nam and B. Han, "Learning multi-domain convolutional neural networks for visual tracking," *arXiv preprint arXiv:1510.07945*, 2015.

[58] H. Nam, M. Baek, and B. Han, "Modeling and propagating CNNs in a tree structure for visual tracking," *arXiv preprint arXiv:1608.07242*, 2016.

[59] L. Bertinetto, J. Valmadre, J. F. Henriques, A. Vedaldi, and P. H. Torr, "Fully-convolutional siamese networks for object tracking," in *European Conference on Computer Vision*. Springer, 2016, pp. 850–865.

[60] G. Zhu, F. Porikli, and H. Li, "Beyond local search: Tracking objects everywhere with instance-specific proposals," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 943–951.

[61] D. Du, H. Qi, L. Wen, Q. Tian, Q. Huang, and S. Lyu, "Geometric hypergraph learning for visual tracking," *IEEE transactions on cybernetics*, 2016.

[62] S. Becker, S. B. Krah, W. Hübner, and M. Arens, "MAD for visual tracker fusion," in *SPIE Security+ Defence*. International Society for Optics and Photonics, 2016, pp. 99 950K–99 950K.

[63] A. F. Rodriguez-Perez, "Maximum margin correlation filters," Ph.D. dissertation, Carnegie Mellon University, 2012.

[64] M. Kristan *et al.*, "VOT challenge," September 2017. [Online]. Available: http://www.votchallenge.net/

[65] Y. Wu, J. Lim, and M. H. Yang, "Object tracking benchmark," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 9, pp. 1834–1848, Sept 2015.

[66] M. Kristan *et al.*, "The visual object tracking VOT2013 challenge results," in *2013 IEEE International Conference on Computer Vision Workshops*, 2013, pp. 98–111.

[67] ——, "The visual object tracking VOT2015 challenge results," in *2015 IEEE International Conference on Computer Vision Workshop (ICCVW)*, 2015, pp. 564–586.

[68] L. Čehovin, A. Leonardis, and M. Kristan, "Visual object tracking performance measures revisited," *IEEE Transactions on Image Processing*, vol. 25, no. 3, pp. 1261–1274, March 2016.

[69] J. A. Fernandez and B. V. K. Kumar, Vijaya Kumar, "Zero-aliasing correlation filters," in *Image and Signal Processing and Analysis (ISPA), 2013 8th International Symposium on*. IEEE, 2013, pp. 101–106.

[70] F. Liu, T. Zhou, K. Fu, and J. Yang, "Robust visual tracking via constrained correlation filter coding," *Pattern Recognition Letters*, vol. 84, pp. 163 – 169, 2016.

[71] B. V. K. Vijaya Kumar, A. Mahalanobis, and D. W. Carlson, "Optimal trade-off synthetic discriminant function filters for arbitrary devices," *Optics Letters*, vol. 19, no. 19, pp. 1556–1558, Oct 1994.

[72] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, pp. 27:1–27:27, 2011, software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[73] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[74] R. Walsh and H. Medeiros, *Detecting Tracking Failures from Correlation Response Maps*. Springer International Publishing, 2016, pp. 125–135.

[75] H. T. Nguyen, M. Worring, and R. van den Boomgaard, "Occlusion robust adaptive template tracking," in *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, vol. 1, 2001, pp. 678–683.

[76] H. T. Nguyen and A. W. M. Smeulders, "Fast occluded object tracking by a robust appearance filter," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 8, pp. 1099–1104, Aug 2004.

[77] F. Pernici and A. Del Bimbo, "Object tracking by oversampling local features," *IEEE transactions on pattern analysis and machine intelligence*, vol. 36, no. 12, pp. 2538–2551, 2014.

[78] S. Siena and B. V. K. Vijaya Kumar, "Detecting occlusion from color information to improve visual tracking," in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, March 2016, pp. 1110–1114.