UNIVERSIDADE FEDERAL DE JUIZ DE FORA
INSTITUTO DE CIÊNCIAS EXATAS
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Helena de Almeida Maia

A Mediator for Multiple Trackers in Long-term Scenario

Juiz de Fora

2016

UNIVERSIDADE FEDERAL DE JUIZ DE FORA
INSTITUTO DE CIÊNCIAS EXATAS
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Helena de Almeida Maia

A Mediator for Multiple Trackers in Long-term Scenario

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação, do Instituto de Ciências Exatas da Universidade Federal de Juiz de Fora como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Orientador: Marcelo Bernardes Vieira

Juiz de Fora

2016

Ficha catalográfica elaborada através do programa de geração
automática da Biblioteca Universitária da UFJF,
com os dados fornecidos pelo(a) autor(a)

Maia, Helena de Almeida.
A Mediator for Multiple Trackers in Long-term Scenario / Helena de Almeida Maia. -- 2016.
66 f. : il.

Orientador: Marcelo Bernardes Vieira
Dissertação (mestrado acadêmico) - Universidade Federal de Juiz de Fora, Instituto de Ciências Exatas. Programa de Pós-Graduação em Ciência da Computação, 2016.

1. Template tracking. 2. Tracking-Learning-Detection. 3. Semisupervised learning. I. Vieira, Marcelo Bernardes, orient. II. Título.

Helena de Almeida Maia

A Mediator for Multiple Trackers in Long-term Scenario

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação, do Instituto de Ciências Exatas da Universidade Federal de Juiz de Fora como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Aprovada em 18 de Março de 2016.

BANCA EXAMINADORA

Prof. D.Sc. Marcelo Bernardes Vieira - Orientador
Universidade Federal de Juiz de Fora

Prof. D.Sc. Rodrigo Luis de Souza da Silva
Universidade Federal de Juiz de Fora

Prof. D.Sc. Hélio Pedrini
Universidade Estadual de Campinas

À minha família.

Aos meus amigos.

AGRADECIMENTOS

Agradeço aos meus pais pelo amor e pelo constante incentivo aos estudos e as minhas irmãs por todo o apoio. Ao Gabriel pela paciência e por motivar a persistir durante esta etapa.

À todos os amigos da UFJF, da graduação e do GCG, que tornaram esse período mais agradável e auxiliaram com revisões e ideias. Aos amigos de Campinas que, mesmo surgindo no final do mestrado, foram importantes por diversos motivos.

Aos professores do DCC que contribuíram para a minha formação. Em especial, ao professor Marcelo Bernardes pela orientação. Ao professor William Schwartz, da UFMG, por suas participações na contextualização e escolha do tema.

Finalmente, agradeço à CAPES e UFJF pelo apoio financeiro, sem o qual esse trabalho não seria possível e a todos que direta ou indiretamente fizeram parte da minha formação.

"If knowledge can create problems, it is not through ignorance that we can solve them." Isaac Asimov

RESUMO

Nos últimos anos, o rastreador TLD (*Tracking-Learning-Detection*) se destacou por combinar um método de rastreamento através do movimento aparente e um método de detecção para o problema de rastreamento de objetos em vídeos. O detector identifica o objeto pelas aparências supostamente confirmadas. O rastreador insere novas aparências no modelo do detector estimando o movimento aparente. A integração das duas respostas é realizada através da mesma métrica de similaridade utilizada pelo detector que pode levar a uma decisão enviesada.

Neste trabalho, é proposto um *framework* para métodos baseados em múltiplos rastreadores onde o componente responsável pela integração das respostas é independente dos rastreadores. Este componente é denominado *mediador*. Seguindo este *framework*, um novo método é proposto para integrar o rastreador por movimento e o detector do rastreador TLD pela combinação das suas estimativas. Os resultados mostram que, quando a integração é independente das métricas de ambos os rastreadores, a performance é melhorada para objetos com significativas variações de aparência durante o vídeo.

Palavras-chave: Rastreamento por template. *Tracking-Learning-Detection.* Aprendizado semisupervisionado.

ABSTRACT

On the problem of tracking objects in videos, a recent and distinguished approach combining tracking and detection methods is the TLD (Tracking-Learning-Detection) framework. The detector identifies the object by its supposedly confirmed appearances. The tracker inserts new appearances into the model using apparent motion. Their outcomes are integrated by using the same similarity metric of the detector which, in our point of view, leads to biased results.

In our work, we propose a framework for generic multitracker methods where the component responsible for the integration is independent from the trackers. We call this component as *mediator*. Using this framework, we propose a new method for integrating the motion tracker and detector from TLD by combining their estimations. Our results show that when the integration is independent of both tracker/detector metrics, the overall tracking is improved for objects with high appearance variations throughout the video.

Keywords: Template tracking. Tracking-Learning-Detection. Semisupervised learning.

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

$\boldsymbol{f}$	Continuous function $\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^m$.
f	Discrete function $f\colon \mathbb{Z}^n \to \mathbb{Z}^m$.
A	Set.
$\|A\|$	Set size.
$\vec{p}$	Point $\vec{p} = (x_1, \cdots, x_n)$.
$\vec{v}$	Vector $\vec{v} = (v_1, \cdots, v_n)$.
$\vec{bb}$	Bounding box coordinates $\vec{bb} = (i_1, j_1, i_2, j_2) \in \mathbb{Z}^4$, with $i_1 < i_2$ and $j_1 < j_2$.
$\lfloor a \rfloor$	The largest integer b such that $b \leq a$.
$\Vert \vec{v} \Vert$	L_2 norm for the vector $\vec{v}$ defined as $\Vert \vec{v} \Vert = \sqrt{\sum v_i^2}$
$\Vert \vec{v} \Vert_1$	L_1 norm for the vector $\vec{v}$ defined as $\Vert \vec{v} \Vert_1 = \sum \vert v_i \vert$
$\langle \vec{v}, \vec{u} \rangle$	Inner product between $\vec{v}$ and $\vec{u}$ defined as $\langle \vec{v}, \vec{u} \rangle = \Vert \vec{v} \Vert \Vert \vec{u} \Vert \cos\theta = \sum v_i u_i$.

LIST OF ACRONYMS

2D Two-dimensional

3D Three-dimensional

ALIEN Appearance Learning In Evidential Nuisance

ALOV Amsterdam Library of Ordinary Videos

LBP Local Binary Pattern

MTA Multihypothesis Trajectory Analysis

NCC Normalized cross-correlation

NN Nearest neighbor

OpenCV Open Source Computer Vision Library

SIFT Scale Invariant Feature Transform

SST Structural Sparse Tracking

SVM Support Vector Machine

TLD Tracking-Learning-Detection

VOT Visual Object Tracking

CONTENTS

1 INTRODUCTION

The central goal of Computer Vision researches is to develop in computers the ability to perceive the environment. Environment perception in human beings is accomplished by extracting cues from sensory inputs. Similarly, Computer Vision researches seek to train computers for extracting and interpreting cues from input signals. With this goal in mind, here we address the problem of Visual Object Tracking (VOT). It consists of estimating object position throughout time using visual cues. The input signal in VOT is a sequence of frames, or a video, and visual cues may be color, depth, apparent motion, among others. The main issue of this problem is how to describe the object and its movement.

Over the past years, we have seen a growing interest in VOT researches due to their diversity of potential applications. Some applications are listed below:

- Automated surveillance: Automated surveillance systems use images from security cameras to detect unusual activities and to prevent accidents or crimes. One of those systems was proposed by Shah et al. (2007), the Knight. By using tracking, Knight is able to follow the object and describe its activity. Since tracking is one step of a major process, it requires high accuracy and low computational cost.

- Augmented reality: It consists of artificially producing new objects over real objects. Sometimes, it requires not only the position, but also the pose of the real object, as the one proposed by Comport et al. (2006).

Other potential applications include human action recognition and human-computer interfaces. The method proposed here is more suitable for applications that do not require object pose.

A common challenge in VOT is the change in the object appearance during the video, caused by illumination variations, pose change, among others. Thus, a good tracker should be trained to find these variety of appearances or adapt to them at runtime. The tracked object may also be covered by other elements, or even be out of the scene. So, the system should be able to recover the object after an occlusion. Recently, we have also seen a growing concern in proposing methods that track farther, i.e., in long sequences. The reason is that the aforementioned situations are more frequent in this scenario.

There is a large variety of trackers in the literature mainly differing in the object modeling and how the trajectories are obtained. Some examples of object representation are articulated models, polygon meshes and appearance examples via images (templates). The first two models are more common when we also want to estimate object pose which is not the case of this work. Here, we focus on the template tracking problem which generally has two approaches to obtain the object trajectory: tracking by motion model (LUCAS; KANADE, 1981; SHI; TOMASI, 1994; MATTHEWS et al., 2004; KALAL et al., 2010b) and by detection (OZUYSAL et al., 2007; PERNICI; BIMBO, 2013).

Tracking by motion model is based on motion estimation methods, mainly optical flow. It consists of optimized searches replacing the template at short intervals. Due to this updating process, the tracker is capable of adapting to new object appearances, but retaining only one template at a time. This becomes an issue when new appearances are not related to the original sample, caused by either gradual or abrupt insertion of background in the template. The error accumulation during tracking is known as the drift problem, which might cause the tracker to permanently lose the appearance of the original object, requiring re-initialization.

In tracking by detection, the object trajectory is obtained through independent detections at each frame. The detector is a classifier that examines the whole frame and decides about object presence or absence in each region. By performing independent detections, it can recover the object after an occlusion using the known appearances. Generally, the detector is trained offline and requires a large set of samples and an exhaustive training or it gets outdated quickly. To deal with this, many recent works propose semisupervised training that collects samples automatically at runtime (PERNICI; BIMBO, 2013; ROSENBERG et al., 2005; KALAL et al., 2012). The key issue of semisupervised learning is the method for sample collection. It must accept new object appearances avoiding noise and other objects which may cover it. By using improper collected samples, the detector might recognize false positives, leading to an effect similar to the drift problem.

In a long-term scenario, both approaches eventually fail. Tracker may lose permanently the object after some time; detector may be outdated quickly. But, what is a failure in one might be a strength in the other. For this reason, Kalal et al. (2012) proposed a semisupervised learning combining both approaches, each representing a different component of their Tracking-Learning-Detection (TLD) framework. It uses the motion model tracker to fill the detector training set with new appearances. The detector corrects tracker failures replacing the template. The complementary nature of the components and the results achieved suggest that this is a promising combination. Not all tracker responses are used for retraining. The detector similarity function is used to validate them and as well to select the system output among tracker and detector responses. In contrast, Rosenberg et al. (2005) show that their detector-independent metric outperforms the detector similarity function in a retraining step because the learning and detection failure cases tend to be distinct.

Here, we define as *mediator* the component responsible for integrating the outcomes and updating multiple trackers. It is composed of one *selector* and many *validators*. The *selector* is the component responsible for receiving all the responses and choosing one to be the system output. Each *validator* is responsible for updating one target tracker. For that, it receives one or more responses from the other trackers and allows or not the updating of the target tracker using these responses. Motivated by Rosenberg et al. (2005), we propose a basic premise for any *mediator*: it should make decisions using their own knowledge and strategy, i.e., work independently of its mediated trackers. This implies that it cannot be strongly based on any of the trackers models.

In this work, we propose a *mediator* for the motion tracker and the detector proposed in TLD framework. The *mediator* in our method is composed of one *selector* and one *validator* that has the detector as target tracker. The system output is used to update the motion tracker without any validation. Differently from TLD, the proposed *mediator* is independently defined.

1.1 PROBLEM DEFINITION

Given a sequence of frames $V = \{f_1, ..., f_m\}$ and a bounding box $\vec{bb}_1$ containing the object of interest in the first frame, we want to estimate the object state in the rest of the sequence. The object state includes its position and scale, if it is visible, given by a bounding box in the respective frame. Thus, we want to find the set of states for the respective frames, i.e., the trajectory $T = \{\vec{bb}_1, ..., \vec{bb}_m\}$, where $\vec{bb}_i$ coordinates are null if the object is not present in frame f_i.

The first bounding box is given by the user and the whole information contained on it represents one object appearance. For adapting to the wide variety of appearances, the system should collect new bounding boxes automatically at runtime, avoiding other objects in the scene. If the object goes out of the scene, the tracker should recover it. Besides, it should work well in long sequences. No restrictions are placed on the tracked object or scenario, except that the bounding box size must be greater than a threshold and the object must be visible in the first frame.

1.2 OBJECTIVES

The primary objective of our work is to propose mediation components that are defined independently of the considered trackers, in order to improve the performance of the original TLD. As secondary objectives, our method is supposed to:

- detect gradual degradation of the motion tracker output for deciding when it is reliable.
- find the soundest estimation between both trackers outputs in each frame.
- not compromise the computational cost when compared with the original method.

1.3 METHODOLOGY

The underlying premise of this work is that the combination of different strategies (i.e., multiple trackers) in a unified method might be more effective to solve the VOT problem than using a single tracker approach. This assumption is well accepted by the scientific community, being a trend in recent researches as Kalal et al. (2012) and Lee et al. (2015). In this scenario, the strategies are indepently applied and the outcomes are combined to get the final response. The central issue of our research is how to combine the outcomes. We call as *mediator* the component responsible for this integration because it mediates between components in potential disagreement. Our first and most important hypothesis is that the more independent from the trackers is the *mediator*, the less unbiased are the results. For us, an independent *mediator* is the one that uses its own strategy and knowledge in order to make fair decisions exploiting the advantages of each mediated tracker. Using a different strategy does not guarantee that the results are fully unbiased. But the failure cases of the trackers and the *mediator* tend to be distinct and so, the *mediator* does not tend to support trackers' failures.

The secondary hypothesis of our work is that the trackers in the combination must be complementary in nature, in the sense that some of them give good results when the other fail. This is desirable to enhance the possibility of having at least one good response available for the *mediator*. If the *mediator* makes decisions through statistical measures, this condition is in fact necessary. When working with the mode, for instance, the use of trackers that fail at the same time might lead the *mediator* to choose the incorrect response.

On this basis, we propose a *mediator* for two complementary trackers from TLD. In our available time, it was not possible to test the addition of other trackers in the framework as it requires a careful analysis about the complementarity of the new tracker. Although we provide some evidences for the validity of our hypothesis based on experiments, further investigations are needed in order to demonstrate it.

The main contribution of this work is an approach that can collect a greater variety of object appearances for detector retraining, if compared to TLD, as the validity decision is not influenced by the detector itself. Similarly, the selection component can identify good responses of the motion tracker, avoiding improper re-initializations. As a result, our tracker outperforms the original, in particular by estimating the correct position more often. In other words, it achieves a wider coverage of the expected trajectory.

1.4 PRIOR DISCUSSIONS

Before proceeding, it is important to bring in some initial issues about our problem. The first issue concerns the appearance concept. For human beings, the appearance comprises the visible aspects of a certain object. These aspects are perceived by us through a very complex process that are related to prior long lived experiences. In this work, we refer as *appearance* the set of statistical measurements taken from the object region in an image, i.e., the information that is available in the region and that the system is capable of extracting. These statistical measurements are called *features*. This way, the problem addressed here is indeed to find regions in frames that present *features* similar to those in the known *appearance*.

The second issue concerns the *features* variability. Several conditions may change the *appearance* of the target object throughout a video. Some examples are lighting changes, occlusions, deformations, different points of view, among others. If the system considers aspects that vary under such conditions, it may fail in identifying the object. There are two basic approaches to the problem: 1. adapting to the variations using the subsequently found regions; 2. consider only the *features* that are strongly invariant in some of those conditions. In addition to the known strong features, the accepted regions may bring new information about the object, but it may also carry undesirable information. Without the supervision of the user, the system has difficulty in differing both information. In our case, the second choice has the advantage of considering only genuine features given in the first frame by the user. Strictly speaking, both approaches cannot solve the indefinitely long tracking problem. So, the goal is in fact to track farther.

In our work, all the framework components must identify the object. Motion tracker and detector collect new *appearances* to track, as originally proposed. For our *selector* we have tested, unsuccessfully, a method for collecting features to validate all trackers responses. Although other methods may be explored, we have chosen to maintain the authenticity of the first original *features* in our selection process. For our *validator*, instead of identifying the object, we have chosen to identify what is not the object, using the object surrounding. If the motion tracker response contains many of these negative features, it is not reliable. As the background comprises many objects that do not share features, collecting new negative features is a more interesting approach than exploring invariant ones.

1.5 OUTLINE

In this chapter, we introduced the problem of object tracking in videos adressed in this work and our proposal. The rest of the work is organized as follows.

In Chapter 2, we present the main fundamentals of our work. Methods for object description are presented in Section 2.1, that includes histograms commonly used in appearance-based tasks, and in Section 2.2, where we briefly describe SIFT (Scale Invariant Feature Tranform) keypoint extraction and matching methods. In Section 2.3, we give details about the TLD tracker and its components that are the basis of our proposal.

In Chapter 3, we present some appearance-based works in literature that attempt to deal with the VOT problem. They are subdivided into two main trends characterized in the beginning of the chapter.

Chapter 4 is dedicated to our method. We first propose a generical *mediator* for multiple trackers and some required properties. From this, we propose a tracker based on TLD. The results of the proposed tracker are presented in Chapter 5. Finally, Chapter 6 presents the conclusions of our work and potential future works.

2 FUNDAMENTALS

In this chapter, we present some fundamental concepts for our work. But, first of all, we introduce some general definitions.

Image: An image is a 2D continuous signal $\boldsymbol{f} : \mathbb{R}^2 \rightarrow \mathbb{R}^c$ which associates a point $\vec{p} = (x, y)$ to a brightness vector $\vec{v} \in \mathbb{R}^c$. Each vector element represents the channel intensity for the point. A digital image f is a discrete representation of an image that is usually seen as an $m \times n$ matrix. In this work, we refer to a grayscale digital image as image and the gray values are in the interval $[0, 255]$. By convention, the image origin is located at the lower left corner.

Video: A video V is a sequence of images ordered by time, $V = \{f_1, \cdots, f_k\}$. Images from a video sequence are often called frames.

Bounding box: A bounding box is a rectangle which possibly contains an object of interest in a particular frame. It provides the position and scale of the object (object state). Formally, it is given by its minimum and maximum coordinates $\vec{bb} = (i_1, j_1, i_2, j_2)$. $\vec{bb} = (0, 0, 0, 0)$ indicates that the object is not visible in the frame.

Trajectory: The object trajectory T is the set of states that it assumes throughout a video V, $T = \{\vec{bb}_1, \cdots, \vec{bb}_k\}$.

2.1 HISTOGRAMS

Histogram is a statistical tool to represent data distribution considering a certain aspect. The set of possible values is divided into intervals (or bins) and each one is associated with the number of observations that falls into it. It is a useful tool for comparing sets of different sizes, since it has a fixed size for all of them and is normalized. Here, we present three histograms that are fundamental for representing object appearances in our proposal. We combine their information to form a rich descriptor similar to Schwartz and Davis (2009). For that, consider f a frame, $f(\vec{p})$ the color intensity for each pixel $\vec{p} = (i, j)$

in f, $\vec{bb}$ the appearance being described and $BB = \{\vec{p} \mid i_1 \leq i \leq i_2, j_1 \leq j \leq j_2\}$ the set of points within $\vec{bb}$.

Color Histogram: This histogram encodes the color distribution for a given image. Different objects tend to have different color distributions. Any color space can be used, but RGB and HSV are more common. To combine the channels brightness, some works compute multiple histograms (one per channel) and concatenate them. In our proposal, we use only one channel. This one-channel histogram is called intensity histogram.

Formally, each element of the color histogram $\vec{c} = (c_1, \cdots, c_{n_c})$ is given by:

$$c_a = \sum_{\vec{p} \in BB} \delta_{a,b(f(\vec{p}))}, \tag{2.1}$$

where $a \in \{1, \cdots, n_c\}$ is the bin index, $\delta_{a,b(f(\vec{p}))}$ is the Kronecker delta and $b(f(\vec{p}))$ is the bin index that the intensity $f(\vec{p})$ fell into. Considering a uniform subdivision of the intensity interval, the bin index is given by $b(f(\vec{p})) = \left\lfloor \frac{f(\vec{p}) \cdot n_c}{255} \right\rfloor + 1$.

Since the spatial locality is not considered, it is a suitable feature vector for objects that present considerable variations during the video (PÉREZ et al., 2002), as it is invariant to rotations, scaling and some deformations. However, not considering position may also lead to a low discriminative power. To overcome this limitation, we combine color information with other visual cues.

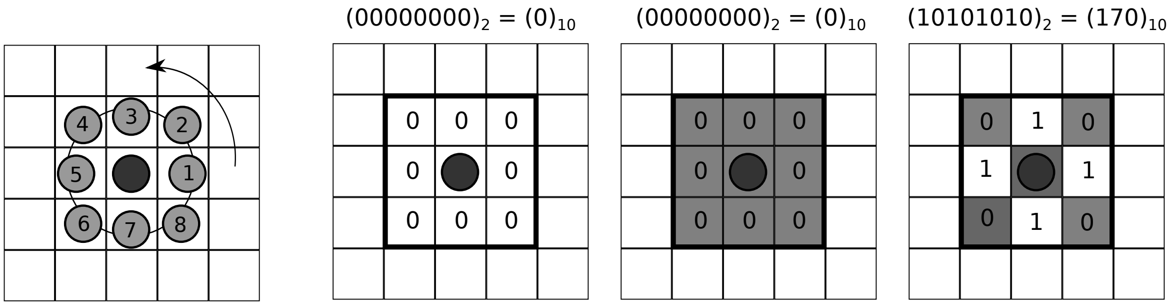
Local Binary Patterns: Local Binary Patterns (LBP) are features that represent local texture of an image and was proposed by Ojala et al. (1996). The LBP for a given point is an integer representing its neighborhood. It is computed through a set of binary tests $BT = \{bt_0, \cdots, bt_7\}$, using the central point $\vec{p}$ intensity as threshold to evaluate each neighbor $\vec{p_u}$:

$$bt_u(\vec{p}) = \begin{cases} 1, & f(\vec{p_u}) > f(\vec{p}) \\ 0, & otherwise \end{cases}, \ u \in \{0, \cdots, 7\}.$$

As the original, we use a 3×3 neighborhood on a unit circle centered by the point $\vec{p}$ (Fig. 2.1(a)), although other neighborhoods may be used. LBP value for the point $\vec{p}$ is given by:

$$lbp(\vec{p}) = \sum_{u \in \{0, \cdots, 7\}} bt_u(\vec{p}) \cdot 2^{7-u}.$$

With 8 neighbors, the maximum value that lbp might assume is $2^8 - 1 = 255$. Examples of LBP value computing are shown in Figure 2.1(b).



(a) (b)

Figure 2.1: Scheme of LBP value computing. The dark gray circle marks the central point $\vec{p}$ whose intensity is used as threshold. (a) General process: binary tests are performed between the threshold and the neighbors intensity in the indicated order. They give 1 for neighbors in lighter color and 0 for neighbors in darker color regarding the threshold. (b) Examples of texture patterns: Homogeneous textures have the LBP value equal to 0 while a different pattern has the value 170.

As color information, LBP values from a given appearance are encoded into histograms. But, for texture the appearance region is subdivided into four non-overlapping blocks and each one generates a different histogram. This way, the set BB is subdivided into BB_1, BB_2, BB_3, BB_4, and each subset BB_i generates a histogram $\vec{l_i} = (l_{i1}, \cdots, l_{in_l})$, where the bins are populated as follows:

$$l_{ia} = \sum_{\vec{p} \in BB_i} \delta_{a,b(lbp(\vec{p}))}. \tag{2.2}$$

The number of bins is $n_l = 2^8 = 256$ and the bin index is given by $b(lbp(\vec{p})) = lbp(\vec{p}) + 1$.

LBP histogram is quite simple and has low computational cost since LBP values might be calculated using binary operations. Different from color histogram, it is not invariant to rotations and scaling. By subdividing the set BB, it considers to some extent texture positions. LBP is invariant to global grayscale changes that preserve intensity orders (monotonic), as global illumination changes.

Histogram of Oriented Gradients: Histogram of oriented gradients (HOG) encodes shape information and the original HOG2D, concerning 2D gradient vectors, was proposed by Dalal and Triggs (2005). In calculus, the gradient vector $\nabla \boldsymbol{f}$ of a multivariable and continuous function $\boldsymbol{f}$ points to the direction of the greatest rate of increase of the function.

Its magnitude indicates how much $\boldsymbol{f}$ rises in that direction. The gradient elements are the partial derivatives at the point $\vec{p}$ with respect to $\boldsymbol{f}$ variables:

$$\nabla\boldsymbol{f}(\vec{p}) = \left(\frac{\partial\boldsymbol{f}(\vec{p})}{\partial x_1}, \cdots, \frac{\partial\boldsymbol{f}(\vec{p})}{\partial x_n}\right) = (G_{x_1}, \cdots, G_{x_n}).$$

For continuous images $\boldsymbol{f} : \mathbb{R}^2 \to \mathbb{R}^c$, high rates of increase indicate significant color variation and, consequently, the presence of edges/borders. When working with a discrete image *f*, the gradient can be approximated by using the Sobel operators:

$$S_i = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \text{ and } S_j = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}.$$

This way, at the point $\vec{p}$, horizontal and vertical derivatives G_i and G_j are given by S_i and S_j convolved with the image *f* around the point.

To compute HOG2D, the gradient vectors $g(\vec{p}) = (G_i, G_j)$ are converted to equivalent polar coordinates $g'(\vec{p}) = (\theta, r)$, with $\theta = \tan^{-1}(\frac{Gj}{Gi})$, $\theta \in [0, 2\pi]$ and $r = \| g(\vec{p}) \|$. The angle θ is used to define the corresponding bins that the gradient vector is assigned for, while its magnitude determines the increase in the bin value. Thus, the shape histogram $\vec{s} = (s_1, \cdots, s_{n_s})$ is populated as follows:

$$s_a = \sum_{\vec{p} \in BB} r(\vec{p})\omega_a(\theta(\vec{p})). \tag{2.3}$$

For HOG, instead of using Kronecker delta, we use a Gaussian function ω_a as a weighting factor, spreading the energy through the neighboring bins. Farthest neighbors receive smaller increases.

As LBP histogram, HOG is not invariant to rotations. It is somewhat invariant to scaling, if the angle frequencies are proportionately kept, and invariant to monotonic grayscale changes.

Cosine similarity for histogram matching: Cosine similarity is a metric for vector comparison. Considering $\vec{v}$ and $\vec{u}$ two vectors in $\mathbb{R}^n$, cosine similarity $\boldsymbol{cos} : \mathbb{R}^n \times \mathbb{R}^n \to [-1, 1]$ is given by:

$$\boldsymbol{cos}(\vec{v}, \vec{u}) = \cos(\theta) = \frac{\langle \vec{v}, \vec{u} \rangle}{\| \vec{v} \| \| \vec{u} \|}. \tag{2.4}$$

By using this metric, vectors with the same orientation get the highest score, while opposite vectors have the smallest score. This is a useful similarity function for histogram comparisons, since it can indicate if two distributions are proportionally similar. As the presented histograms represent the frequencies of the observations, all of their coordinates are positive. Thus, the similarity between two histograms falls in the range $[0, 1]$.

2.2 SIFT AND OCCLUSION DETECTION

Lowe (2004) proposed a method for extracting, describing and matching invariant features from images for object recognition. This method is called Scale Invariant Feature Transform (SIFT) since it represents the whole object/image by scale invariant keypoints. Besides being invariant to scaling, SIFT keypoints are invariant to rotation and translation, partially invariant to change in illumination and robust to changes in 3D viewpoint and noise addition.

The first step of the method is to detect prominent points at different scales using the scale space representation. In the scale space, the original image is represented by a family of smoothed versions of itself, parametrized by the level of smoothing. At higher scale levels, fine details are filtered out, which means that they are not discriminative in that scale. Points with low contrast and along edges are discarded. The remaining keypoints are described by the gradient of the points from its surrounding region. The gradient vectors are encoded into 16 orientation histograms with 8 bins, that are concatenated to form the keypoint descriptor $\vec{d} \in \mathbb{R}^{128}$.

Object recognition is carried out by means of individual matchings of the features. Every candidate keypoint is compared with stored keypoints from a training data using the Euclidean distance. The nearest neighbors are the most similar keypoints. To avoid poor correspondences, it filters some matchings using a threshold. However, instead of applying the threshold only in the nearest neighbor, it is applied in the ratio between the nearest and the second nearest keypoint in the keypoints set. That is, considering $\boldsymbol{1nn}(\vec{d})$ and $\boldsymbol{2nn}(\vec{d})$ the nearest neighbors of $\vec{d}$ in the keypoints set C, $\vec{d}$ is accepted if $\frac{\|\vec{d}-\boldsymbol{1nn}(\vec{d})\|}{\|\vec{d}-\boldsymbol{2nn}(\vec{d})\|} < \lambda_c$. This way, it discards features that have a good match with many keypoints (non-discriminative), or that do not have a good match with any keypoints, keeping the cases where $\| \vec{d} - \boldsymbol{1nn}(\vec{d}) \| \ll \| \vec{d} - \boldsymbol{2nn}(\vec{d}) \|$.

Exploring the discriminative property of the SIFT keypoints, Pernici and Bimbo (2013) proposed an occlusion detector for retraining decisions in their ALIEN tracker. For each frame, the tracker extracts keypoints in a search window around the last position. Current object position is estimated through weak alignment of SIFT keypoints using stored features of the object and its context. This results in a set *OBB* for the frame f_t containing estimated object keypoints. These keypoints may include features that matched with the context. The key idea to detect occlusions is to find how many of them exist in *OBB*. From the estimation process this is straightforward: occlusion features are keypoints that matched with the context and are in *OBB*. If the number of occlusion features are greater than a threshold, the estimation is considered unreliable and the sets of keypoints remains unaltered. Otherwise, it stores the matched keypoints in the sets. Object and occlusion features are kept in the set until it reaches a maximum size. Context features, on the other hand, are kept for a fixed number of frames. Here, we adapt this occlusion detection method for our motion tracker validation, to detect if its estimation is reliable for the discriminative tracker retraining.

2.3 TLD FRAMEWORK

In this section, we present the original components of TLD tracker. Three of them are kept in our proposal: the median flow tracker, the cascaded detector and the P-N learning. We also present the original *validator* and *selector*, although with other names. It is important to consider that these names, as well the *mediator*, were defined here to represent our substitute methods as independent components.

2.3.1 MEDIAN FLOW TRACKER

The motion tracker used in the TLD framework is a slightly modified version of the median flow tracker (KALAL et al., 2010b). In TLD, it has an additional method to detect failures. At each time t, the motion tracker receives as input two frames, previous f_{t-1} (reference) and current f_t (search space), and a bounding box $\vec{bb}_{t-1}$ representing the template. It assumes that the bounding box contains an appearance of the target object, even though this is not always true. One of the *mediator*'s responsibilities is to provide valid bounding boxes for the motion tracker. Using the appearance indicated by $\vec{bb}_{t-1}$ in f_{t-1}, the motion tracker estimates object position in the current frame, exchanging a bounding box $\vec{bb}_t$. The steps for each time t is presented in Algorithm 1.

Algorithm 1: Median Flow Tracker.

```
Data: Previous frame $f_{t-1}$, current frame $f_t$, previous bounding box $\vec{bb}_{t-1}$.
Result: Estimated bounding box $\vec{bb}_t$.
begin
    $G \leftarrow$ grid($f_{t-1}$, $\vec{bb}_{t-1}$);
    $G' \leftarrow$ pyramidalTracker($f_{t-1}$,$f_t$,$G$);
    ($G$, $G'$) $\leftarrow$ pointFiltering($G$,$G'$);
    $\vec{bb}_t \leftarrow$ medianFlow($G$,$G'$,$\vec{bb}_{t-1}$);
end
```

Median flow tracker estimates the object motion by tracking template points selected in the function *grid()*. The displacements of these points are estimated in the function *pyramidalTracker()* and unreliable displacements are discarded in the function *pointFiltering()*. Overall motion is estimated in the function *medianFlow()* using the remaining points displacements. Failures are detected in this function as well. The above functions are described in more details in the following paragraphs.

1. **Grid:** In this function, the motion tracker selects equally spaced points within $\vec{bb}_{t-1}$, resulting in a 10×10 grid G. New points are selected at each frame. Displacement of the selected points are estimated in the pyramidal tracker method.

2. **Pyramidal tracker:** For point tracking, the median flow tracker uses a version of the Lucas-Kanade method (LUCAS; KANADE, 1981). In this method, for a certain point $\vec{p} = (i, j)$ we want to find the displacement vector $\vec{d} = (d_i, d_j)$ such that the neighborhood of $\vec{p}$ in f_{t-1} is similar to the neighborhood of $\vec{p} + \vec{d}$ in f_t. Each neighborhood is defined as a $(2w + 1) \times (2w + 1)$ window around $\vec{p}$. The error measure between two points is given by the L_2 norm. Thus, for the whole window we are looking for a vector $\vec{d}$ that minimizes the error function given by:

$$\boldsymbol{e}(\vec{d}) = \sum_{\vec{q}=(i',j')} [f_{t-1}(\vec{q}) - f_t(\vec{q} + \vec{d})]^2,$$

where $i - w \leq i' \leq i + w$ and $j - w \leq j' \leq j + w$, i.e., $\vec{q}$ is in the neighborhood of $\vec{p}$. This method assumes small and homogeneous displacement within the window. Lower w values capture fine motions, while are robust to noise.

Median flow tracker uses the pyramidal version proposed by Bouguet (2001). Lucas-Kanade and its pyramidal version estimate a vector field for a set of points, which in median flow case is the set G. In Algorithm 1, the function *pyramidalTracker()* outputs the set G' of terminal points of each vector. Then, we must select the most reliable points to estimate object motion. The following paragraph presents two approaches to extract these points.

3. **Point filtering:** In this stage, two error functions are used to evaluate the quality of the displacement vectors estimated by the pyramidal tracker. They are presented below.

    (a) **Normalized cross-correlation:** Normalized cross-correlation (NCC) is a similarity measure between two signals and it is widely used for template matching. NCC gives the probability of a target template being located at a certain position of an image. NCC has low sensitivity to absolute intensity changes between reference and target images due to normalization.

    For point filtering, an error measure is computed from NCC value. Considering the point $\vec{p} = (i_1, j_1) \in G$, its corresponding point $\vec{q} = (i_2, j_2) \in G$*'* and two $2w_n + 1 \times 2w_n + 1$ windows arround them, the NCC value is given by:

$$NCC(\vec{p}, \vec{q}) = \frac{\sum_{\vec{d}=(i,j))}(f_{t-1}(\vec{p}+\vec{d}) - \bar{w}_1) \cdot (f_t(\vec{q}+\vec{d}) - \bar{w}_2)}{\sqrt{\sum_{\vec{d}=(i,j))}(f_{t-1}(\vec{p}+\vec{d}) - \bar{w}_1)^2 \cdot \sum_{\vec{d}=(i,j)}(f_t(\vec{q}+\vec{d}) - \bar{w}_2)^2}},$$

    where $-w_n \leq i \leq w_n$, $-w_n \leq j \leq w_n$ and $\bar{w}_1$ and $\bar{w}_2$ are the mean intensity in each window. It falls in the range $[-1, 1]$, where the higher the NCC value, the better the matching. From this, the NCC error is defined as:

$$e_{NCC}(\vec{p}) = 0.5(1 - NCC(\vec{p}, \vec{q})),$$

    which falls in the range $[0, 1]$. Lower NCC error means higher similarity and, consequently, higher point reliability. 50% of the points with the worst values are eliminated.

    (b) **Forward-Backward error:** Forward-backward (FB) is a dissimilarity function proposed by Kalal et al. (2010b). It is based on the assumption that the point tracker satisfies the symmetry property for equivalence relations. That is, if the tracker relates a point $\vec{a}$ to a point $\vec{b}$, it is expected that it relates the point $\vec{b}$ to the point $\vec{a}$ again. In fact, if the backward point lies on the neighborhood of $\vec{a}$, the point $\vec{a}$ can be seen as reliable. Formally, let $\vec{p} = (i, j)$ be a point in the first image. By applying pyramidal tracking forward, we have the corresponding point $\vec{p}_F = (i_F, j_F)$ in the second image. Now, by applying the tracking backward using $\vec{p}_F$ (i.e. from the second to the first image), we have the point $\vec{p}_B = (i_B, j_B)$. FB error for the point $\vec{p}$ is given by the Euclidean distance between $\vec{p}$ and $\vec{p}_B$:

$$e_{FB}(\vec{p}) = \parallel \vec{p}_B - \vec{p} \parallel .$$

FB error is depicted in Figure 2.2. Similar to NCC, 50% of the points with the worst error are eliminated. By filtering out bad points, tracker avoids occluded points and noises for the next stage.

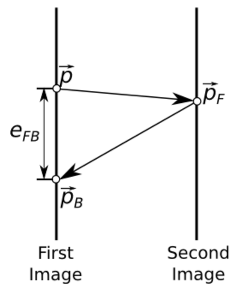


Figure 2.2: FB error for point a $\vec{p}$. Adapted from Kalal et al. (2010b).

4. **Median flow:** Point tracker gives us a set of vectors which represent the motion. After filtering out bad points, we can estimate the object position using the median flow method proposed by Kalal et al. (2010b). The bounding box displacement is the median in each coordinate $\vec{d}_m = (d_{im}, d_{jm})$. This way, the method is robust to impulse noise and are able to generalize the movement. For the scale factor, we compute the ratio of the distance between each pair of points in the first and second image. That is, let $\vec{p}_1, \vec{p}_2 \in G$ be two points in the first image and $\vec{q}_1, \vec{q}_2 \in G'$ their corresponding points in the second image. The distance ratio is given by:

$$dist_{ratio}(\vec{p}_1, \vec{p}_2) = \frac{\parallel \vec{p}_2 - \vec{p}_1 \parallel}{\parallel \vec{q}_2 - \vec{q}_1 \parallel}.$$

The bounding box scale factor s_m is the median of $dist_{ratio}$ for all the pair of points. Note that, the bounding box aspect ratio will be kept since we have the same scale factor for both coordinates. Therefore, given a bounding box $\vec{bb}_{t-1}$, the displacement vector $\vec{d_m^{t-1}} = (d_{im}^{t-1}, d_{im}^{t-1})$ and the scale factor s_m^{t-1}, the coordinates of the new bounding box $\vec{bb}_t$ are calculated from:

$$\begin{cases} i_1^t = i_1^{t-1} + d_{xm}^{t-1} + \frac{w^{t-1}}{2}(1 - s_m^i) \\ j_1^t = j_1^{t-1} + d_{ym}^{t-1} + \frac{h^{t-1}}{2}(1 - s_m^i) \\ i_2^t = i_2^{t-1} + d_{xm}^{t-1} - \frac{w^{t-1}}{2}(1 - s_m^i) \\ j_2^t = j_2^{t-1} + d_{ym}^{t-1} - \frac{h^{t-1}}{2}(1 - s_m^i) \end{cases},$$

where $w^{t-1} = i_2^{t-1} - i_1^{t-1} + 1$ and $h^{t-1} = j_2^{t-1} - j_1^{t-1} + 1$ are the width and height of $\vec{bb}_{t-1}$ respectively. The scale is applied around the bounding box center and the center position is kept by this transformation. TLD median flow has an additional failure detector. They define the residual displacement for each point $\vec{p_i} \in G$ as being $r(\vec{p_i}) = \| \vec{d_i} - \vec{d_m} \|_1$. If the median of r is greater than 10, a failure is detected and the motion tracker declares the object as not visible.

2.3.2 CASCADED DETECTOR AND P-N LEARNING

The detector in the TLD framework is inspired on Viola and Jones (2001) and Ozuysal et al. (2007) methods. The steps for each time t are presented in Algorithm 2. For every frame, the detector generates many windows (function *scanningWindows()*) to decide about the presence or absence of the object in each one. These windows pass through a three-stage cascaded classifier that filters out many of them in each stage. The stages are represented by the functions *varianceFilter()*, *ensembleClassifier()*, *nearestNeighborClassifier()*. At the end of the process, the windows that have not been discarded are more likely to contain the object. Details of the functions are given in the following paragraphs.

1. **Scanning windows:** The cascaded detector performs a global search in the frames. For that, it generates windows in many sizes and at different positions, keeping the aspect ratio of the first bounding box. Each window is a candidate to be the tracked object. To decide the most confident one, they pass through the cascaded classifier.

2. **Variance filter:** The first stage blocks windows that do not present a minimum

Algorithm 2: Cascaded detector.

```
Data: Current frame $f_t$.
Result: Estimated bounding boxes $W_t$.
begin
    $W_1 \leftarrow$ scanningWindows($f_t$);
    $W_2 \leftarrow$ varianceFilter($f_t, W_1$);
    $W_3 \leftarrow$ ensembleClassifier($f_t, W_2$);
    $W \leftarrow$ nearestNeighborClassifier($f_t, W_3$);
end
```

level of details considering the first appearance. This is accomplished through variance values. This stage has low computational cost since the variance may be quickly computed by using integral images (VIOLA; JONES, 2001). Windows with less than half of the first variance value are discarded. No retraining is performed in this stage.

3. **Ensemble classifier:** In the second stage, the windows are evaluated through intensity comparisons between pairs of points. The ensemble is a set of classifiers that evaluate the remaining candidate windows. Each member of the ensemble performs a set of tests in the candidate window different from the ones of the other members. The pairs of points for the tests are selected once for each member and remain the same till the end of the tracking. Similar to LBP, this set of tests results in an integer x, representing the window pattern extracted by the member i. The member vote for the candidate window is given by the posterior probability $\boldsymbol{p}_i(y = 1 \mid x)$, computed in the training/retraining step. The window passes to the next stage if the average vote of the members is greater than 0.5.

   In the training and retraining steps, the P-N learning component provides a set of positive and negative samples for the ensemble. Each member of the ensemble uses these samples to initialize/update its posteriors probabilities. The posteriors are given by $\boldsymbol{p}_i(y = 1 \mid x) = \frac{\#p_i}{\#p_i + \#n_i}$, where $\#p_i$ is the number of positive samples that present the pattern x and $\#n_i$ is the number of negative samples that present the same pattern by the member i. Before being counted in $\#p_i$ or $\#n_i$, each sample is classified by the ensemble. If the classification is incorrect, the corresponding members' counters are increased.

4. **Nearest neighbor classifier (NN classifier):** In the last stage, the detector

generates a similarity score for each window considering an object model. The object model comprises two sets of image samples collected so far, one for positive and one for negative examples. For each candidate, the classifier searches for the nearest neighbors in these sets by using NCC measure. Let s^+ be the NCC similarity related to the positive nearest neighbor and s^- related to the negative. The similarity score for a candidate window is given by the relative similarity $s_r = \frac{s^+}{s^+ + s^-}$. This candidate is considered positive if its relative similarity is greater than a threshold, $s_r > \theta$. The windows that pass through this last stage are the output of the detector. Note that, the detector may accept more than one window as being positive.

P-N learning component also provides a set of positive and negative samples for the NN classifier in the training and retraining steps. Each of them is classified by NN and, if the classification is incorrect, the samples are inserted in the object model.

P-N learning: P-N learning is the component responsible to send new samples for the cascaded classifier. For that, at each frame it uses a motion tracker response approved by the validation method. This component is based on two structural constraints. They analyze the whole set of scanning windows to send new samples to the ensemble and the current NN output to send back to it. P-constraint assumes that the object moves along a trajectory estimated by the motion tracker. Windows labeled as negative that are close to the valid response of the motion tracker are considered false negatives. N-constraint assumes that the object is located in only one region at each frame. So, windows far from the valid response labeled as positive are considered false positives. Their respective false positives and negatives are sent to the ensemble and NN classifiers.

2.3.3 SELECTION AND VALIDATION

Original selection and validation methods are based on a measure very similar to the relative similarity from the NN classifier. The difference is that instead of using the whole set of positive samples to find the nearest neighbor it uses only the first half of the samples, the oldest ones. This measure is called conservative similarity s_c. Besides using a similar measure, it also uses the object model from NN classifier.

The selection method, originally called integration, receives as input the detector and motion tracker outputs. It computes the conservative similarity for each response. If the detector outputs a window far from the motion tracker estimation and with a higher similarity value, this is chosen to be the final response. This final response is also used to re-initialize the tracker, i.e., to replace its template.

The validation method, originally being the core of reliability, evaluates the motion tracker responses to send to the learning component. There are two conditions under which the motion tracker response is considered valid. The first case is when its conservative similarity is greater than a threshold. The second one is if the motion tracker was not re-initialized since the last valid response. This second case makes the validation accept more appearances estimated by the motion tracker. But it is important to remember that the selection method decides when the tracker may be re-initialized.

3 RELATED WORKS

In this chapter, we present some related works that attempt to deal with the aforementioned challenges in VOT. VOT is a hard problem subject to several conditions. For this reason, there is a large variety of solutions and categorizing them is not trivial. As said in Smeulders et al. (2014) survey, the methods do not seem to fit solely in one line of thought. But, one may note two main trends to solve the problem (KALAL et al., 2012; SMEULDERS et al., 2014), even though many works mix the concepts. The first group comprises the trackers by matching or trackers by motion model. The second group comprises the discriminative trackers or trackers by detection.

Trackers by matching estimate the interframe object motion through template matching. They explore the spatiotemporal coherence of the motion, performing local search around the last position found. They use only one template for the matching and, as a consequence of this and the restricted search, they are very fast. To deal with the appearance variety, many tracker replaces the template by the discovered appearance. However, as a side-effect, the estimation can bring background information, affecting future estimations. This leads to the major problem of the approach, the drift problem. Once an incorrect estimation replaces the template, the original may be never recovered. This is typically caused by fast motion, blurring or occlusions. Being aware of this limitation, many works focus on trackers that resist longer. Other nontraditional trackers propose the use of an extended model, to overcome the lack of memory.

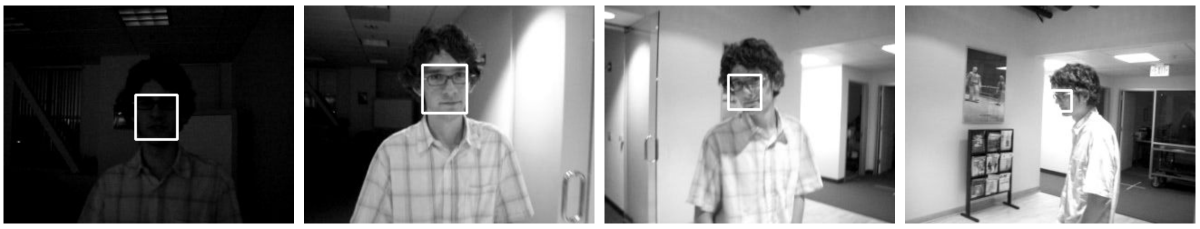


Figure 3.1: Motion tracker adaptability depicted in David sequence from TLD dataset. By replacing the template, it can handle changes in pose and illumination.

Discriminative trackers perform global searches and treat every frame independently. They build classifiers based on foreground-background distinction and so, are able to detect if the object is out of the scene. Generally, they train the binary classifier in

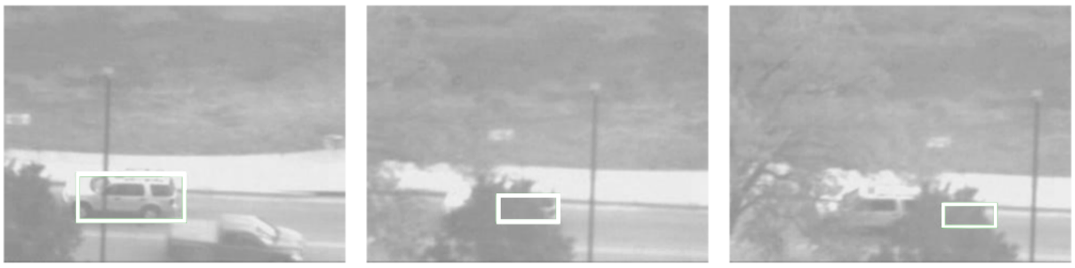


Figure 3.2: Example of drift failure in Car sequence from TLD dataset using a motion tracker. As the template was replaced, the tracker cannot recover the car after an occlusion.

a supervised fashion. Since the tracker should work well for many objects and their appearance changes during the video, the training stage has to be exhausting or otherwise, they become outdated rapidly. Moreover, this exhausting supervised training requires many labeled examples, which is not always the case of the tracking scenario. As we said, the object is indicated by only one appearance in the first frame. Some works produce artificially new examples by deforming the first appearance, but they still cannot overcome some situations. Finally, the classification cost in discriminative trackers is greater than in the traditional motion trackers.

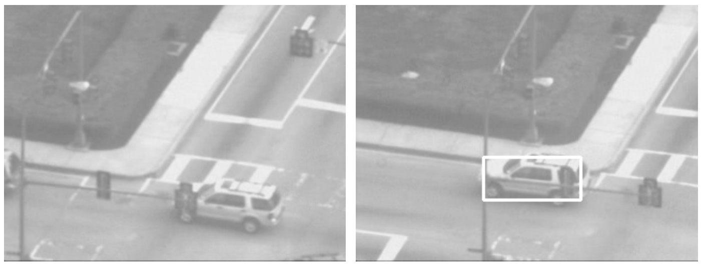


Figure 3.3: Example of object recovering after an occlusion in Car sequence from TLD dataset using a discriminative tracker.

A summary table of both approaches is shown in Table 3.1. Novel proposals seem to have been converging toward the idea that mixing the concepts improves both trends. For instance, some trackers by matching keeps an extended model, which are more common in tracking by detection, to avoid the drift problem. We also have seen a growth interesting in detectors that collects new samples at runtime (semisupervised training). In Section 3.1 and 3.2, we give details about some trackers in the literature.

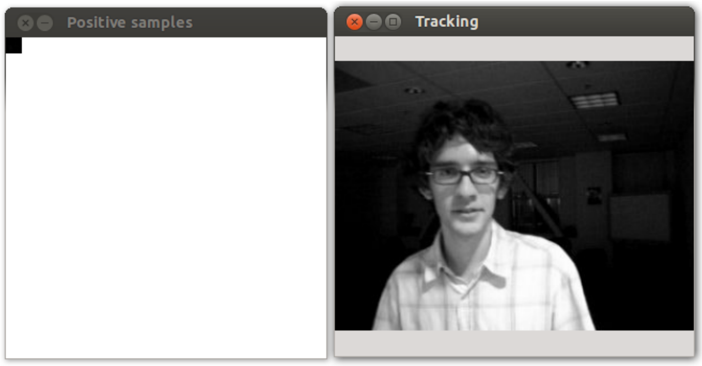


Figure 3.4: A discriminative tracker (detector) in David sequence from TLD dataset. After some frames, the first appearance stored in the positive set is useless and the detector is not able to deal with the appearance changing.

	By matching	**Discriminative**
Reference	Reduced positive model	Positive and negative models
Search	Local	Global
Strength	High speed and adaptability (Fig. 3.1)	Recollection (Fig. 3.3)
Weakness	Drift problem (Fig. 3.2)	Becomes outdated rapidly (Fig. 3.4)
Requires	Reference correction	Exhaustive supervised training or online retraining

Table 3.1: Summary of the two main trends for object tracking.

3.1 TRACKERS BY MATCHING

A remarkable motion tracker is the Lucas-Kanade (LUCAS; KANADE, 1981; BAKER; MATTHEWS, 2004; BOUGUET, 2001). It estimates the object motion by estimating individual point displacements between two consecutive frames, the called point tracking. By estimating point translation, it can handle other linear transformations on the object appearance. The template is taking from the previous frame and, as it uses points from this new template without criteria, it may drift rapidly. From that, new methods based on Lucas-Kanade tracker emerged trying to solve the problem of rapid drifting, tracking farther.

One of those was proposed by Shi and Tomasi (1994). Besides storing the template from the previous frame, this tracker stores the first one. The interframe motion is estimated as in Lucas-Kanade tracker, but the first template is used to assess the quality of the feature points being tracked. If a feature point become too dissimilar from the original, it is permanently abandoned. This prevents a background feature in the template from being considered for tracking. However, it does not select new features and the tracking process may stop owing to lack of points. Another feature selection method was proposed by Kalal et al. (2010b) for their median flow tracker. They proposed the forward-backward error to filter the displacement vectors estimated with Lucas-Kanade tracker. The remaining vectors are used to estimate the object motion. But, different from Shi and Tomasi (1994), new points are extracted at each new template.

A different approach was proposed by Matthews et al. (2004). As in Shi and Tomasi (1994), it stores the initial template for assessment and another one for matching. But, in this tracker the first one is used to decide whether the matching template can be replaced. If the current estimation is sufficiently similar to the first one, it becomes the next template. Otherwise, the old one is kept. The difference is that there is always a template to track.

Even tracking farther than Lucas-Kanade tracker, all of the works presented above eventually fail because they cannot recover the object. This occurs because they have only one template for matching, as the first one is used only for assessment. An incorrect template leads to the drift problem in the same way.

To overcome the lack of memory of the traditional trackers by matching, some authors proposed trackers by matching with extended model or generative trackers. An example is the IVT tracker (Incremental Visual Tracking) that stores more than one previous templates, proposed by Ross et al. (2008). They define a contribution factor that is higher for the latest templates. After a predefined number of frames, old templates are abandoned. But, the method also fails in partial and full occlusion that continue for a long period. The tracking method in the IVT is based on particle filters which is very common as Lucas-Kanade method in motion trackers. Another generative motion tracker exploring particle filter was proposed by Zhang et al. (2015), the SST tracker (Structural Sparse Tracking). It belongs to a subclass of the generative motion trackers: the sparse trackers. Sparse trackers represent the candidates as a sparse linear combination of the stored templates. In SST, this is done by using local patches inside the candidate and templates, preserving the spatial structure. The main advantage of using local patches is that the tracker is able to overcome partial occlusions.

3.2 DISCRIMINATIVE TRACKERS

An important work for object detection was proposed by Viola and Jones (2001) using positive and negative examples of the object. The detector scans the whole frame using sliding windows in different scales, but keeping the same aspect ratio. At each position, it decides whether the object is present or not. The classification process is subdivided into stages using a cascaded scheme. At each stage, several candidates are filtered out. The first stage has a lower computational cost, but it is applied in a larger amount of windows. This scheme reduces the computational cost of evaluating a large amount of candidate windows all at once. The cascaded classifier is trained once offline.

More recent works explore semisupervised training to reduce the cost of the training step. For instance, Rosenberg et al. (2005) propose a method for detector retraining. The detector is trained with a small set of labeled samples. At each classification, it collects more samples among its own responses. It chooses samples with the highest score based on a selection metric. They test two selection metrics: the same used for detector classification and an independently defined metric. Since the goal is to improve the detector knowledge, it does not seem reasonably use its own metric. Indeed, the independently defined metric outperforms the detector metric. They also show that the results achieved by the detector trained in this manner are comparable with those achieved using a supervised training.

For updating its detector, Kalal et al. (2010a) propose the P-N learning which is the basis for the TLD tracker (KALAL et al., 2012). It consists of using a motion tracker to judge detector responses based on two structural constraints. By the first constraint, responses near from the motion tracker estimation that received a negative label are considered misclassified. This constraint is called P-constraint since it recognizes positive samples. The second constraint is the N-constraint that detects false positives. It lies in the fact that the object is located in only one region at each frame. False negatives and positives are inserted in the detector training set. As the motion tracker can fail, its responses are evaluated before being considered for retraining. As said before, this evaluation is based on the detector metric. Note that the goal is to use the motion tracker to insert appearances that the detector is not able to recognize by itself in its training set. But, the outdated knowledge of the detector is used to decide whether these appearances may be considered. Here, we argue that evaluating these samples with an external metric improves the detector.

Another approach was proposed by (PERNICI; BIMBO, 2013), the ALIEN tracker (Appearance Learning In Evidential Nuisance). The ALIEN tracker achieved impressive results in the TLD dataset by exploring SIFT keypoints (Scale Invariant Features Transform). Keypoints are extracted from the first appearance to form the positive set and from the region around it, the context, to form the negative set. The object position is estimated by keypoint alignment. If the estimation is considered reliable, the matched keypoints are collected. The estimation is considered reliable when the number of context keypoints that match with keypoints from the object is smaller than a threshold, i.e., no occlusion has occurred. Since the drift problem is strongly related to background insertion in the object region, this is a suitable technique for our *validator*. Further details about the occlusion detection are given in Section 2.2.

Lee et al. (2015) proposed the MTA tracker (Multihypothesis Trajectory Analysis). It consists of three discriminative trackers working together. The classifiers are based on SVM (Support Vector Machine) and each one works on a different aspect of the object: texture, color and illumination invariant features. The trackers estimate three forward and backward trajectories for a fixed frame interval ending in the current one. They propose a robustness score over the three pairs of trajectories to select one. The forward trajectory from the winner classifier is used to retrain the others. Although we use different components, this method fits in our general framework: multiple trackers and a *mediator* composed of a *selector* that uses an external metric.

4 PROPOSED METHOD

Motivated by the possibility of combining different trackers in a unified method, we propose a general framework depicted in Figure 4.1. It is composed of n trackers working together. Each one estimates the object position using its own model and strategy. As in an ensemble of classifiers, a good combination of trackers would be that in which individual accuracies are satisfactory and they make independent errors (HANSEN; SALAMON, 1990). Thus, in the ideal scenario, at least one tracker per frame would estimate a correct response.

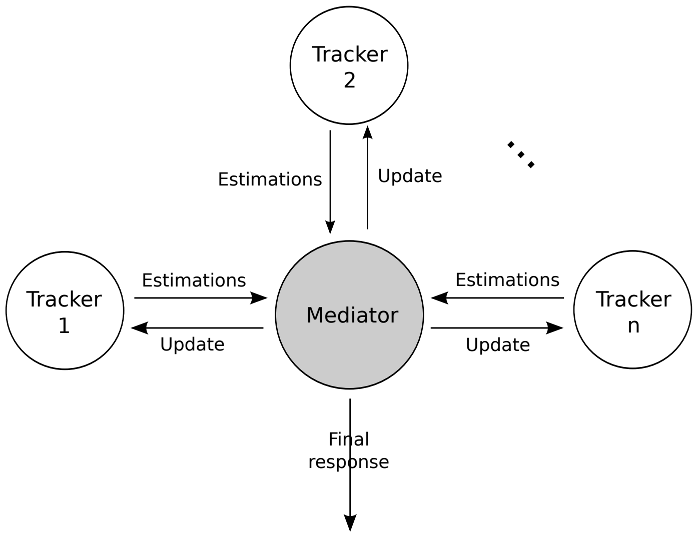


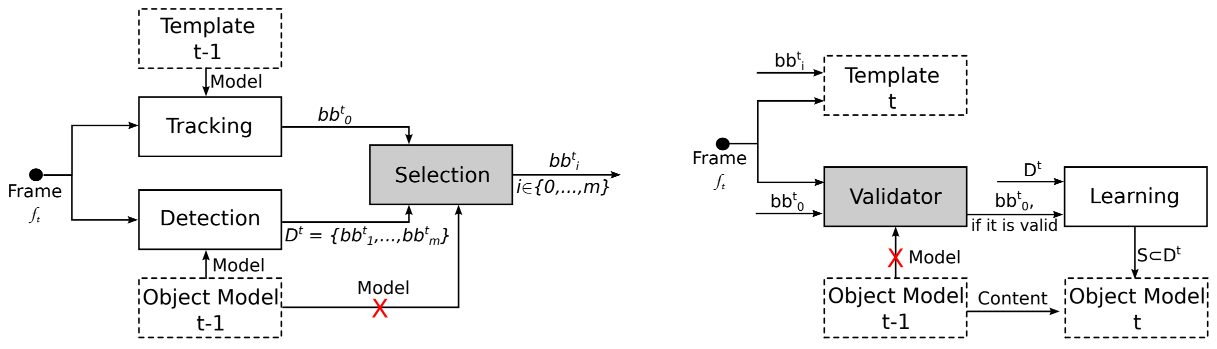
Figure 4.1: General framework for multiple trackers. The *mediator* component selects the final response and corrects the trackers.

The component responsible for integrating the outcomes is the *mediator*. It receives all the responses and chooses one to be the system output. It does not produce any new response. If all the input responses are incorrect, its output is going to be incorrect too. In our approach, the mediator assumes that one of the trackers is giving a good response for each frame on average. The rationale behind this assumption is that if all trackers are good for a given application, the *mediator* is useless. The same applies if all the trackers are also bad estimators anywhere in the video. Its strenght lies in the possibility of, among multiple tracker strategies, on average at least one will perform fairly well when favorable video conditions appear for its tracking expertise. For each frame, its responsibility is thus to find the correct response among a group of estimations. This is done by a subcomponent that we call the *selector*. As an external and central agent, the *mediator* also has new information to update the trackers. So, it is desirable that the *mediator* provides some feedbacks for its mediated trackers considering the other responses for the frame. The subcomponents responsible for this task are the *validators*, each one having a target tracker to update.

If the *mediator* is based on the knowledge of one of its mediated trackers, it tends to support this tracker's failures and spread the error to the others. So, we propose a basic premise for any *mediator*: it should make decisions using their own knowledge and strategy, i.e., work independently of its mediated trackers. In the ideal scenario, it should be fully unbiased, but this is somewhat a strong requirement because, essentially, most features are based on the same information.

4.1 METHOD OVERVIEW

Following the presented framework, we propose a tracker based on TLD. Our method contains two trackers kept from the TLD: the median flow tracker and the cascaded detector with the P-N learning detailed in Chapter 2. The *mediator* is composed of one *selector* and one *validator* that has the detector as target tracker. The system output is used to update the motion tracker without any validation. We aim to show that our independently-defined selection and validation methods outperform the original.



(a) Response selection. (b) Trackers update.

Figure 4.2: Overview of our method where the *validator* and *selector* make decisions only with their own knowledge. The red crosses indicate connections of the original TLD removed in our proposal.

An overview of our method is depicted in Figure 4.2. System samples and responses are sub-images delimited by bounding boxes possibly containing the object being tracked. Thus, all components only exchange BB coordinates containing the sub-image in the respective frame. The reference appearance is a bounding box in the first frame given by user to initialize the motion tracker and train the detector. The template model is used exclusively by the tracker and contains a reference frame (the previous one) and its bounding box. The object model comprises a list of sample bounding boxes collected so far (training set) and two similarity functions for candidate matching: the relative similarity used by the NN classifier that considers the whole positive set, and conservative similarity used by the original selection and validation methods that considers only the first half of the positive set.

The *selector* gives the estimated object bounding box for each frame f_t (Fig. 4.2(a)). For that, it receives one bounding box estimated from the motion tracker and a list of bounding boxes estimated as likely containing the object from the detector. Then, it selects one of the inputs, if any, to be the system response. The selected response is also used to update the motion tracker template (Fig. 4.2(b)). The *validator* decides if the tracker estimation can be used for retraining the detector. If current tracker estimation is considered invalid, the object model remains the same. Otherwise, the P-N learning component uses it as reference to retrain the cascaded detector. The detector responses considered incorrect are sent to the model to improve future detections.

In the original method, the *selector* component uses the object model to choose the final answer. We argue that it tends to benefit detector responses because they are based on the same similarity functions and samples. It also affects future tracker estimations since the final response replaces the old template. In the *validator*, the reliability score of the tracker response is also given by the object model. This way, the *validator* uses the detector knowledge to decide when the detector could be retrained. In our point of view, the *selector* and *validator* components have to be defined independently of the detector and motion tracker in order to get the best of each approach. So, we propose new validation and selection methods presented in the following subsections.

4.1.1 PROPOSED SELECTOR

The *selector* receives one bounding box from the tracker and a list of bounding boxes from the detector. Tracker and detector might not always provide bounding boxes, meaning the object is not visible. If no bounding boxes are given, the *selector* has no output either, i.e., object not found. Otherwise, it selects one of the inputs as the system response.

Using the first sample, the *selector* has to decide which response presents a sound appearance of the tracked object at each frame. But the object can assume different appearances throughout the video and a good *selector* should take them into account to accept the most likely and reject the least. To address the problem, we use a rich descriptor combining color, texture and shape, similar to Schwartz and Davis (2009).

Color information is represented by an intensity histogram $\vec{c} \in \mathbb{R}^{256}$ (Eq. 2.1) using one channel and 256 bins ($n_c = 256$). Texture is represented by four LBP histograms $\vec{l_1}, \vec{l_2}, \vec{l_3}, \vec{l_4} \in \mathbb{R}^{256}$ (Eq. 2.2), resulting in 1024 features. Shape is described using a histogram of oriented gradient $\vec{s} \in \mathbb{R}^{16}$ (Eq. 2.3). Each histogram is normalized using L_2 norm. The descriptor is a set $H = \{\vec{c}, \vec{l_1}, \vec{l_2}, \vec{l_3}, \vec{l_4}, \vec{s}\}$ that contains these features. We store the descriptor *H'* of the initial appearance and compute new descriptors H_j for each response sent to the *selector* at runtime (Alg. 3).

Algorithm 3: Response selection.

```
Data: Current frame $f_t$, tracker responses $\{\vec{bb}_0^t\} \cup \{\vec{bb}_1^t, \cdots \vec{bb}_m^t\}$.
Result: Selected response $\vec{bb}_i^t$.
begin
    $maxSim \leftarrow 0$ ;
    foreach $\vec{bb}_j^t$ do
        //Compute candidate descriptor
        $\vec{c} \leftarrow$ colorHistogram($f_t$, $\vec{bb}_j^t$);
        $\{\vec{l_1}, \vec{l_2}, \vec{l_3}, \vec{l_4}\} \leftarrow$ LBPhistograms($f_t$, $\vec{bb}_j^t$);
        $\vec{s} \leftarrow$ HOG($f_t$, $\vec{bb}_j^t$);
        $H_j \leftarrow \{\vec{c}, \vec{l_1}, \vec{l_2}, \vec{l_3}, \vec{l_4}, \vec{s}\}$;

        //Candidate similarity
        $\boldsymbol{s}_c(H_j) \leftarrow$ colorSimilarity($H'$, $H_j$);
        $\boldsymbol{s}_l(H_j) \leftarrow$ textureSimilarity($H'$, $H_j$);
        $\boldsymbol{s}_s(H_j) \leftarrow$ shapeSimilarity($H'$, $H_j$);
        $\boldsymbol{s}(H_j) \leftarrow \frac{s_c(H_j)+s_l(H_j)+s_s(H_j)}{3}$;

        if $\boldsymbol{s}(H_j) > maxSim$ then
            $maxSim \leftarrow \boldsymbol{s}(H_j)$;
            $i \leftarrow j$;
        end if
    end foreach
end
```

The goal of the *selector* is to pick the largest similarity response to the first descriptor. Since each aspect is represented by different number of histograms and/or features, we compute three individual similarity values whose votes are evenly weighted to generate the final value. For color and shape, the vote is given by the cosine similarity (Eq. 2.4) between the corresponding histograms from the first descriptor and the candidate, i.e., $\boldsymbol{s}_c(H) = \mathbf{cos}(\vec{c}, \vec{c'})$ and $\boldsymbol{s}_s(H) = \mathbf{cos}(\vec{s}, \vec{s'})$. For texture, the vote is the average of the four LBP histograms:

$$\boldsymbol{s}_l(H) = \frac{\mathbf{cos}(\vec{l_1}, \vec{l'}_1) + \mathbf{cos}(\vec{l_2}, \vec{l'}_2) + \mathbf{cos}(\vec{l_3}, \vec{l'}_3) + \mathbf{cos}(\vec{l_4}, \vec{l'}_4)}{4}.$$

The final similarity $\boldsymbol{s}(H)$ is the average between the votes

$$\boldsymbol{s}(H) = \frac{\boldsymbol{s}_c(H) + \boldsymbol{s}_l(H) + \boldsymbol{s}_s(H)}{3}.$$

As introduced in the Chapter 2, the chosen histograms are invariant under different conditions, in an uncorrelated manner among each other. When the object assumes an appearance that affect one of them, it is possible that the other similarities achieve higher values. By using an evenly weighted mean, we reduce the effect of adverse conditions in the final similarity.

4.1.2 PROPOSED VALIDATOR

Trackers by motion model generally fail when dealing with insertion of background in the template. TLD's motion tracker includes a *failure detector* to identify abrupt changes such as occlusions, but gradual changes still represent a challenge considering the tracker's lack of memory. Since it performs local searches, its responses are somewhat close to the last one given. For this reason, template degradation often come from elements in the object neighborhood. To prevent corrupted samples from being added to the detector training set, we keep a record of context features as proposed by Pernici and Bimbo (PERNICI; BIMBO, 2013). The context is a region surrounding the object's bounding boxes within a fixed margin.

In the first frame f_0, we extract SIFT points from the context, defined by a margin m, forming the initial set C_0. At each new frame f_t, we match feature points extracted from the motion tracker's result $\vec{bb}_t^0$ with the previous context features C_{t-1} (Alg. 4). The matching follows the same steps as proposed in Lowe (2004). $\boldsymbol{1nn}(\vec{d})$ and $\boldsymbol{2nn}(\vec{d})$ are, respectively, the first and second nearest neighbors of the descriptor $\vec{d}$ in C_{t-1}. If the number of points matched with the context exceeds a threshold n_o, the tracked sub-image

Algorithm 4: Motion tracker validation

```
Data: Current frame $f_t$, motion tracker response $\vec{bb}_t^0 = (i_1, i_1, j_2, j_2)$.
Result: Response validity $v$ (true or false).
begin
    $v \leftarrow false$;
    //Extract SIFT keypoints from object
    $P \leftarrow \{(i,j) \in \mathbb{Z}^2 \mid i_1 \leq i \leq i_2, j_1 \leq j \leq j_2\}$;
    $S_t \leftarrow \{(\vec{p},\vec{d}) \mid \vec{p} \in P$ is a SIFT keypoint, $\vec{d} \in \mathbb{R}^{128}$ is its descriptor$\}$;
    //Compute matching features with $C_{t-1}$
    $C_t^* \leftarrow \{(\vec{p},\vec{d}) \in S_t \mid \frac{\|\vec{d} - \mathit{1nn}(\vec{d})\|}{\|\vec{d} - \mathit{2nn}(\vec{d})\|} < \lambda_c\}$;
    if $|C_t^*| \leq n_o$ then
        //Reliable response
        $v \leftarrow true$;
        //Extract SIFT keypoints from current context
        $P' \leftarrow \{(i,j) \in \mathbb{Z}^2 \setminus P \mid i_1 - m \leq i \leq i_2 + m, j_1 - m \leq j \leq j_2 + m\}$
        $S'_t \leftarrow \{(\vec{p},\vec{d}) \mid \vec{p} \in P'$ is a SIFT keypoint, $\vec{d} \in \mathbb{R}^{128}$ is its descriptor$\}$
        //Store occlusion features
        $D_t \leftarrow D_{t-1} \cup C_t^*$
        if $|D_t| > n_d$ then $RandomRemoval(D_t, n_d)$
        //Context set update
        $C_t = D_t \cup (\bigcup_{\tau=t-l}^{t} S'_\tau)$
    end if
end
```

is not reliable. Otherwise, $\vec{bb}_t^0$ is valid and used for detector retraining (Fig. 4.2(b)). Context features S'_t are collected around $\vec{bb}_t^0$, within the margin m, for future matchings. They are kept for $l > 1$ frames. The occlusion features D_t formed by accumulated matched points, on the other hand, are kept indefinitely to make sure they will not be detected as object after l frames. When $|D_t|$ reaches a maximum size, some features are randomly removed to control set growth. Thus, the new context set C_t is the union of D_t and the l most recent context features $\{S'_\tau \mid \tau > t - l\}$.

5 EXPERIMENTAL RESULTS

In this chapter, we present the results of our method. In the first two sections, we introduce the datasets (Sec. 5.1.1) and the evaluation protocol (Sec. 5.2) for our experiments. The experiments results are shown in the remainder of the chapter.

5.1 DATASETS

As trackers, there is a large variety of datasets in the literature. Here, we choose two for evaluating our method. The first one is the TLD dataset (KALAL et al., 2012), as the main goal is to compare our method with TLD tracker. The second one is the ALOV300++ (Amsterdam Library of Ordinary Videos) proposed by Smeulders et al. (2014). We are going to give more details about them in following subsections.

5.1.1 TLD DATASET

In Tracking-Learning-Detection paper, Kalal et al. (2012) proposed a new dataset to evaluate long-term trackers (Fig. 5.1). This dataset consists of 10 sequences of different objects under challenging conditions as shown in Table 5.1. They combined six existing short sequences with four additional long sequences. Many sequences of this dataset are monochromatic and/or have low resolution (about 320 × 240).

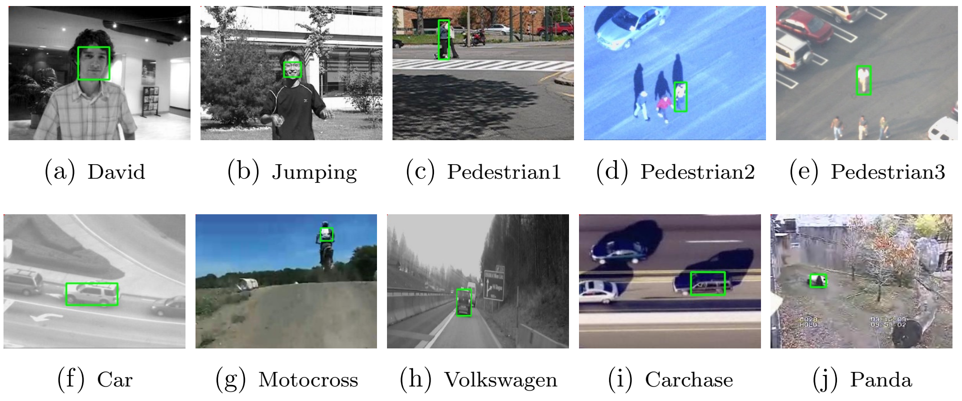

(a) David (b) Jumping (c) Pedestrian1 (d) Pedestrian2 (e) Pedestrian3
(f) Car (g) Motocross (h) Volkswagen (i) Carchase (j) Panda

Figure 5.1: TLD sequences (KALAL et al., 2012).

Besides providing the sequences, they provided the initial object position and the expected answers, or ground truth, for each sequence. More than 50% of occlusion and more than 90° of out-of-plane rotation are considered that the object is not visible. This is indicated by the ground truth. Tracker performance is given by the precision, recall and mainly by f-measure. More details about tracker evaluation are given in section 5.2.

Name	Frames	Mov. camera	Partial occ.	Full occ.	Pose change	Illum. change	Scale change	Similar objects
1. David	761	yes	yes	no	yes	yes	yes	no
2. Jumping	313	yes	no	no	no	no	no	no
3. Pedestrian 1	140	yes	no	no	no	no	no	no
4. Pedestrian 2	338	yes	yes	yes	no	no	no	yes
5. Pedestrian 3	184	yes	yes	yes	no	no	no	yes
6. Car	945	yes	yes	yes	no	no	no	yes
7. Motocross	2665	yes	yes	yes	yes	yes	yes	yes
8. Volkswagen	8576	yes	yes	yes	yes	yes	yes	yes
9. Carchase	9928	yes	yes	yes	yes	yes	yes	yes
10. Panda	3000	yes	yes	yes	yes	yes	yes	no

Table 5.1: TLD dataset (KALAL et al., 2012).

5.1.2 ALOV300++ DATASET

Smeulders et al. (2014) provided an extensive experimental survey about the VOT problem. They argued that the existing datasets have a reduced number of sequences and are limited to one application or scenario. So, they proposed a new dataset with 314 sequences to cover diverse circumstances as possible. Among them, there are some common sequences from other tracking datasets. The sequences are subdivided into fourteen classes. Examples of class sequences are given in Figure 5.2. The average length of the sequences in the thirteen short-term classes is 9.2 seconds with a maximum of 35 seconds. In the long-term class, the length is between one and two minutes. The sequences contain color images in many resolutions ranging from 192×144 to 3840×2160.

As TLD, the ALOV300++ dataset provides the ground truth. The expected answers are annotated by a rectangular bounding box every fifth frame. For rapid motion, the annotations are more frequent. The intermediate annotations are given by linear interpolation. For performance analysis, they proposed the use of survival curves, presented in Section 5.2.

5.2 EVALUATION PROTOCOL - TRAJECTORY QUALITY

For evaluating a tracking method in a particular sequence, we have to evaluate the accuracy of its generated trajectory $T = \{\vec{bb}_1, \cdots, \vec{bb}_k\}$. Tracking datasets usually provides

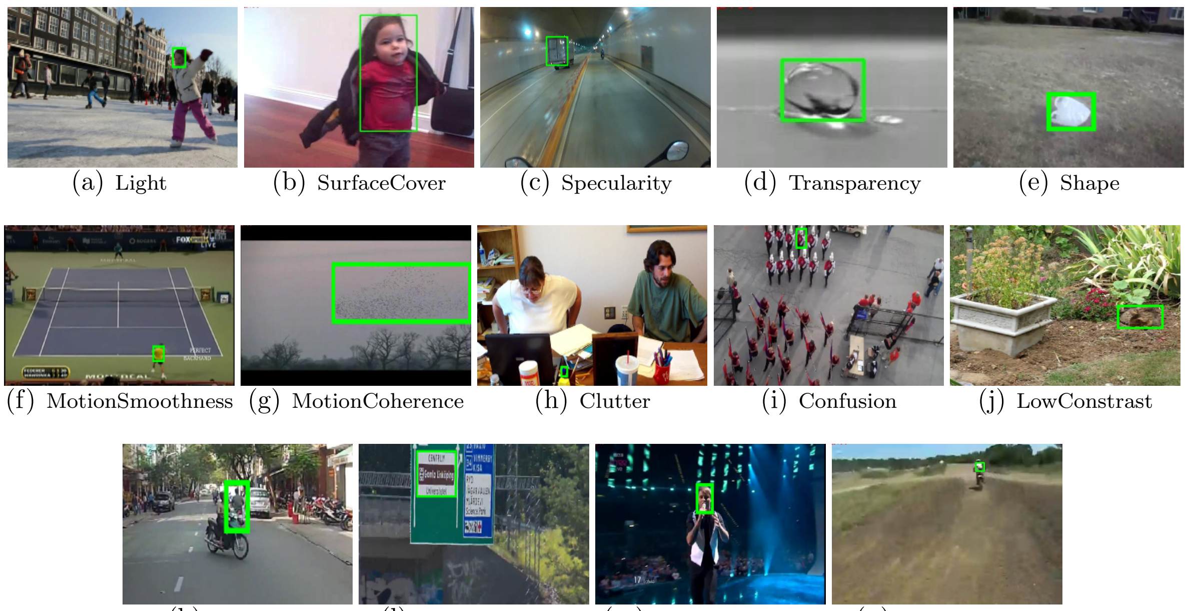

(a) Light (b) SurfaceCover (c) Specularity (d) Transparency (e) Shape
(f) MotionSmoothness (g) MotionCoherence (h) Clutter (i) Confusion (j) LowConstrast
(k) Occlusion (l) MovingCamera (m) ZoomingCamera (n) LongDuration

Figure 5.2: ALOV300++ examples (SMEULDERS et al., 2014).

the expected trajectory $T^* = \{\vec{bb}_1^*, \cdots, \vec{bb}_k^*\}$, where each bounding box $\vec{bb}_t^*$ delimits the object position or indicates that it is not visible in the frame f_t. A tracker response $\vec{bb}_t$ is evaluated through the overlap value with the corresponding ground truth $\vec{bb}_t^*$. The overlap value between two visible bounding boxes is given by the ratio between their intersection and union, i.e., $\boldsymbol{overlap}(\vec{bb}_t, \vec{bb}_t^*) = \frac{\vec{bb}_t \cap \vec{bb}_t^*}{\vec{bb}_t \cup \vec{bb}_t^*}$ (Fig. 5.3). The response is considered correct if both the response and corresponding ground truth are visible, and the overlap value is greater than a threshold (25% or 50% depending on the tracker being compared). Higher overlap values mean that the tracker response better fits the ground truth.

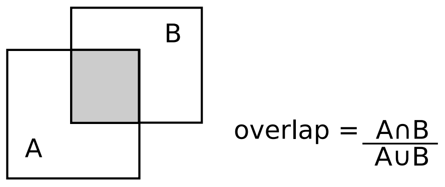


Figure 5.3: Overlap value for two bounding boxes.

For the whole trajectory, the perfomance is given by precision, recall and f-measure (f-score). Precision P is the number of correct responses divided by the number of the responses declared visible by the tracker. Recall R is the number of correct responses divided by the number of visible ground truths. F-measure F is given by $F = \frac{2PR}{P+R}$ and it is the main metric for comparing results, since it combines precision and recall. Tracker overall performance for the whole dataset is given by the mean performance weighted by the number of frames of the sequence. This evaluation protocol is valid for both datasets used in this work. Following the provided protocols, we use an overlap threshold equal to 25% for tests in the TLD dataset and 50% for ALOV300++ tests.

Survival curve: The reduced number of videos of TLD allows individual analysis as presented above. For ALOV300++ dataset, on the other hand, the analysis is carried out via class perfomance. For that, the f-scores for a given class are sorted from the best to the worst, generating a survival curve. The resulting curve indicates how many sequences and to what percentage of each sequence the tracker survives. If the tracker has a great performance for a few sequences, but not for all, the curve falls fast. Class perfomance is also given by the mean performance weighted by its sequence's number of frames.

5.3 RESULTS

In our experiments, we compare the original and the proposed *selector*/*validator*. The components were combined into four variations: original *selector* and *validator* (*os-ov*), original *selector* and proposed *validator* (*os-pv*), proposed *selector* and original *validator* (*ps-ov*), proposed *selector* and *validator* (*ps-pv*). For every variation, we tested different tracker and detector settings such as different window sizes for the tracker optical flow. In the following results, we used the setting with the best overall performance for each variation. For the *validator*, we use $n_o = 2$, $m = 20$, $n_d = 1500$ and $l = 10$ as proposed in the ALIEN (PERNICI; BIMBO, 2013) and $\lambda_c = 0.7$ that empirically gave the best results. The best combination of parameters and components was tested only on TLD dataset, for simplicity, and kept in the ALOV300++.

We fully implemented the TLD tracker to conduct these experiments. By using our own implementation instead of the published results, we insure that the achieved results was caused by the component replacement. All experiments were performed in a machine with an Intel® Core™i7, 2.93GHz processor, 8GB of memory and Ubuntu 13.10 operating system running a single thread. The method was implemented in C++ using the OpenCV (Open Source Computer Vision) library to handle video capture and manipulation. We also used the pyramidal version of Lucas-Kanade method for optical flow estimation and the Sobel operators to compute the gradients for the HOG descriptor, both provided by the same library. For SIFT keypoint extraction, we used the VLfeat library.

5.3.1 SELECTOR GENERALITY POWER

The goal of this first experiment is to investigate the selection metric quality. As said, the *selector* assumes that at least one of the input response is correct and their task is to recognize it. This way, it must give a high similarity score for the correct response. Considering that, we have tested the *selectors* on the expected bounding boxes, generating a similarity curve. This curve gives us a clue about the generality power of each *selector*. As original selection uses collected samples, we compute the similarity score for the expected bounding boxes during the tracking process.

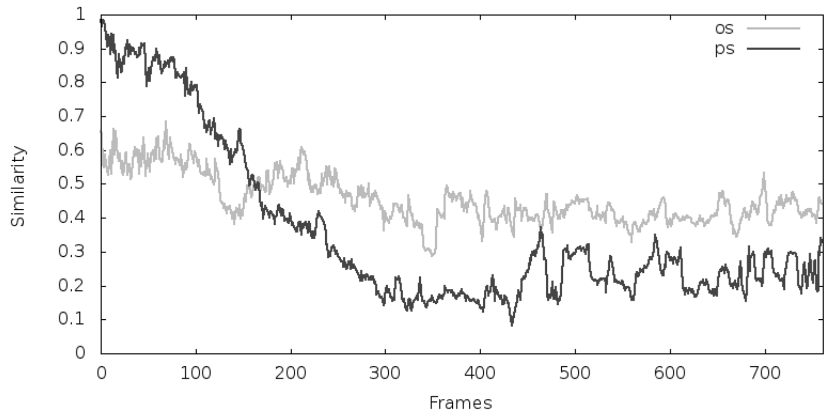


Figure 5.4: Similarity values for the David ground truth using the original (*os*) and proposed (*ps*) selection similarities. The higher the values, the better the generality power of the similarity function.

The proposed selection outperforms the original one in most of the sequences, except for David sequence (Fig. 5.4). Our method fails because of lack of details in the first frame of the David sequence (Fig. 5.5). Since we work with partially invariant features and David sequence presents a drastically change in illumination, the other features cannot handle the color similarity decrease. By collecting new samples, the original method is able to obtain better scores in this sequence.

For all of the other sequences, our method achieved better scores. In some cases it was slightly better: Pedestrian2, Pedestrian3 (Fig. 5.6), Volkswagen and Panda, and in the others it was significantly better: Jumping, Pedestrian1, Car (Fig. 5.7), Motocross and Carchase. This first experiment indicates a greater generality power of our method, which may be an initial clue for component replacement. In the following experiment, we test the effect of the *selector* replacement on tracker performance.

Figure 5.5: First frame of David sequence. The lack of details affects our *selector* metric.

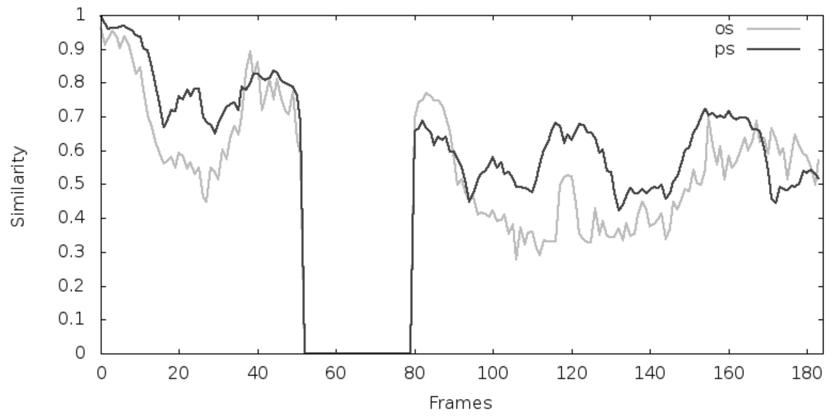


Figure 5.6: Similarity values for the Pedestrian3 ground truth using the original (os) and proposed (ps) selection similarities. The higher the values, the better the generality power of the similarity function.

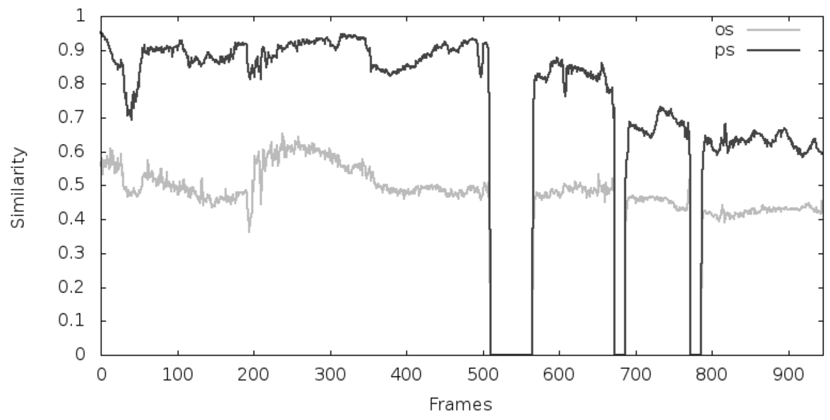


Figure 5.7: Similarity values for the Car ground truth using the original (os) and proposed (ps) selection similarities. The higher the values, the better the generality power of the similarity function.

5.3.2 COMPONENT REPLACEMENT

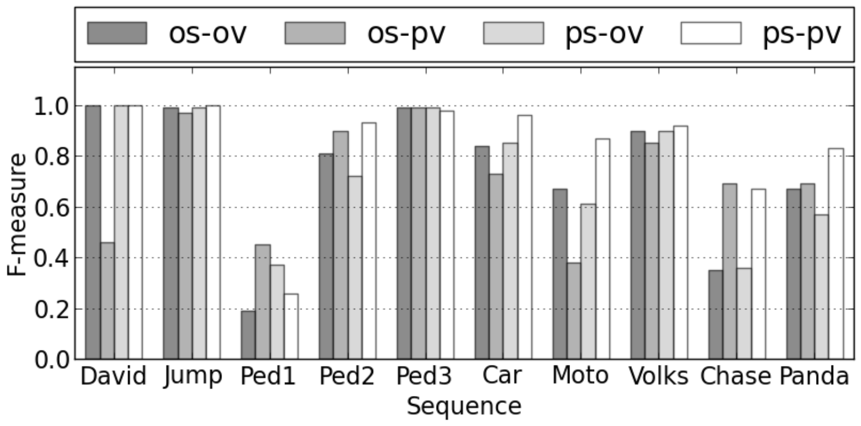


Figure 5.8: F-measure by sequence and variation. *os* is the original *selector* and *ps* the proposed one. *ov* is the original *validator* and *pv* the proposed one.

In the second experiment, we analyze the effect of replacing each component in the original method using the aforementioned variations. F-scores by sequence and variation are shown in Figure 5.8. For some sequences (David, Jumping, Car, Motocross, Volkswagen) the f-measure decreases considerably when replacing only the *validator* (*os-pv*). A possible cause of this low performance is incorrect motion tracker re-initialization by the *selector*. Figure 5.9 shows an example during the David sequence, where the *selector* picks the detector's response even when the tracker gives a better one. This leads to a template change in the tracker, affecting its future estimations and, consequently, the *validator*'s tasks. This occurs in other sequences and illustrates how the error from a component may cascade through the system if the failure cases are the same (the detector and *selector* have failed). As such, if the *validator* cannot accept good samples for detector retraining, the detector cannot send good responses to the *selector* either. We argue that all modules must take their decisions independently to reduce this coupling problem.

The performance for *ps-ov* remains the same for some sequences (David, Jumping, Pedestrian3, Volkswagen) and increased for others (Pedestrian1, Car, Carchase). In Pedestrian2, Motocross and Panda, on the other hand, the performance has decreased. This may be caused by poor quality of trackers responses or poor *selector* performance in these sequences. But, note that even in the sequences where the performance decreased by replacing one of the components, using both of our decoupled proposals (*ps-pv*) give better results related to *os-ov*.

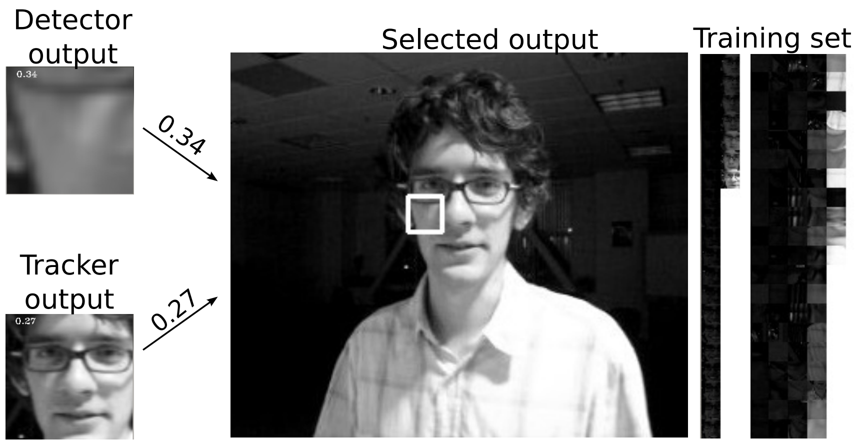


Figure 5.9: Example of incorrect tracker re-initialization in David sequence using *os-pv*. Even with a good training set, the similarity value for the correct tracker output is smaller. It is important to remember that *os* uses only the first 50% of the samples.

Additional variations: We have also tested an additional validation method based on our selection metric (*pv2*). In this method, we follow the same steps of original validation, but instead of using detector conservative similarity and model we use the mean cosine similarity from the *selector*. So, *pv* is also independent from the detector. With this method, we generated two additional variations *os-pv2* and *ps-pv2*, covering all the possible combinations with the available *selectors* and *validators*. Overall performance for the six combinations are shown in Table 5.2. They are ordered from the worst f-measure to the best. We can see that changing *ov* to one of our independent *validator* was important for recall improvement, even keeping *os*, which means that the tracker covered more of the expected trajectory. This may occur because the *validator* accepts more samples for detector retraining and so, detector is able to accept different views from the object. In situations as full occlusions, the motion tracker inevitably fails. But the updated detector is able to recover the object.

In the best combination, besides the *selector* and *validator* being independent from the detector, they are also independent from each other. As those component may fail, its failures cases must be distinct too. We want to highlight in these experiments that, by the strong connection between the components, we have to avoid reinforcement of failures.

5.3.3 SELECTOR RETRAINING

Our proposed *selector* uses only the first appearance to judge the input responses. But we have also tested sample collection throughout the video for this component. In this

Variation	Overall perfomance P / R / F
ps-ov	0.85 / 0.58 / 0.63
os-ov	0.88 / 0.59 / 0.64
os-pv	0.73 / 0.70 / 0.71
os-pv2	0.84 / 0.70 / 0.73
ps-pv2	0.82 / 0.72 / 0.75
ps-pv	0.92 / 0.76 / 0.81

Table 5.2: Comparison among overall performance of the six variations. They are ordered from the worst f-measure to the best.

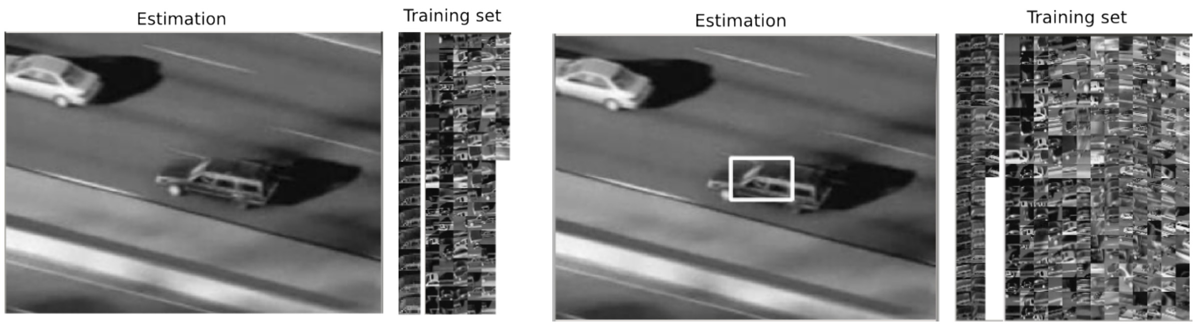
experiment the *selector* stores the selected response for future selections. Candidate bounding boxes sent to the *selector* are compared with each stored samples following the same steps when using only one. However, we use these individual scores to compute a model score. We have tested three basic variants for this:

- Highest: the model score is given by the highest score.
- Median: the model score is given by the median of the scores.
- Mean: the model score is given by the mean score.

Sequence	Frames	Highest P / R / F	Median P / R / F	Mean P / R / F
David	761	**1.00 / 1.00 / 1.00**	**1.00 / 1.00 / 1.00**	**1.00 / 1.00 / 1.00**
Jumping	313	**1.00 / 1.00 / 1.00**	**1.00 / 1.00 / 1.00**	**1.00 / 1.00 / 1.00**
Pedestrian 1	140	**0.62 / 0.26 / 0.37**	0.37 / 0.21 / 0.26	0.37 / 0.21 / 0.26
Pedestrian 2	338	0.96 / 0.83 / 0.89	**0.90 / 0.97 / 0.93**	**0.90 / 0.97 / 0.93**
Pedestrian 3	184	**0.98 / 1.00 / 0.99**	0.96 / 1.00 / 0.98	0.96 / 1.00 / 0.98
Car	945	**0.94 / 0.99 / 0.96**	**0.93 / 0.99 / 0.96**	**0.93 / 0.99 / 0.96**
Motocross	2665	0.68 / 0.54 / 0.60	0.74 / 0.85 / 0.79	**0.87 / 0.77 / 0.82**
Volkswagen	8576	0.64 / 0.71 / 0.68	0.75 / 0.88 / 0.81	**0.81 / 0.95 / 0.87**
Carchase	9928	**0.72 / 0.50 / 0.59**	0.64 / 0.50 / 0.56	**0.66 / 0.54 / 0.59**
Panda	3000	0.05 / 0.05 / 0.05	**0.47 / 0.52 / 0.50**	0.28 / 0.29 / 0.29
mean	26850	0.64 / 0.57 / 0.59	0.70 / 0.70 / 0.69	**0.71 / 0.71 / 0.71**

Table 5.3: Comparison among different model similarities when collecting samples for the *selector* in *ps-pv*. Model similarities are ordered by the highest score, the median of the scores or the mean score.

The results for these three model scores are shown in Table 5.3 using *ps-pv*. We can see that the mean similarity achieved the best results. However, neither of the three options outperformed the *selector* without multiple samples. Being a classifier, the *selector* would need an external metric for retraining/correction as the detector and motion tracker.



(a) *os-ov* (b) *ps-pv*

Figure 5.10: Comparison between *os-ov* and *ps-pv* perfomances for a given frame in *Carchase* sequence. *ps-pv* estimated the correct position because it collected a greater variety of appearances, different from *os-ov* that failed in this frame.

But, proposing a metric or a component to retrain the *selector* would make the problem recursive, considering that it is already a component for mediation. So, we chose to keep our *selector* working only with invariant features from the first appearance.

5.3.4 *OS-OV* × *PS-PV* - DETAILED RESULTS

Detailed results of *os-ov* and *ps-pv* are given in Table 5.4. Note that our proposal increased the recall of almost all the sequences without compromising the precision. This means that our method is capable of tracking farther. In particular, sequences *Motocross*, *Carchase* and *Panda* had a meaningful improvement. The objects in *Motocross* and *Carchase* change their pose throughout the sequence. In *Panda*, the object deforms constantly. This variety of appearances becomes a failure case of *os-ov* since its *validator*, by specifically using normalized cross-correlation (NCC), does not include new appearances that are very far from the known ones. By choosing NCC, *os-ov* imbues to its *validator* a property from the detector that is not found in the tracker. This goes against the main goal of using the tracker's property of accepting new appearances to improve the detector. In *ps-pv*, by assigning to the *validator* a less partial method, it is possible to obtain a richer training set, that accepts more distinct appearances. We atribute this *validator* property, along with proper *selector* activity, to the increase in performance. Figure 5.10 shows the object state estimated by each of the trackers in *Carchase* sequence. We can see that *ps-pv* obtained a richer training set, allowing the correct estimation of the object position when it changes its pose, while *os-ov* improperly declares the object as not visible in the same frame.

Video	Frames	*os-ov* (TLD) *P* / *R* / *F*	*ps-pv* (our) *P* / *R* / *F*
David	761	**1.00 / 1.00 / 1.00**	**1.00 / 1.00 / 1.00**
Jumping	313	1.00 / 0.98 / 0.99	**1.00 / 1.00 / 1.00**
Pedestrian1	140	0.39 / 0.13 / 0.19	0.37 / 0.21 / 0.26
Pedestrian2	338	0.99 / 0.68 / 0.81	**0.90 / 0.97 / 0.93**
Pedestrian3	184	**0.99 / 1.00 / 0.99**	0.96 / 1.00 / 0.98
Car	945	0.87 / 0.81 / 0.84	0.93 / 0.99 / 0.96
Motocross	2665	0.78 / 0.58 / 0.67	**0.88 / 0.86 / 0.87**
Volkswagen	8576	0.88 / 0.92 / 0.90	0.93 / 0.91 / 0.92
Carchase	9928	0.96 / 0.21 / 0.35	0.95 / 0.51 / 0.67
Panda	3000	0.64 / 0.70 / 0.67	**0.80 / 0.87 / 0.83**
Mean	26850	0.88 / 0.59 / 0.64	0.92 / 0.76 / 0.81

Table 5.4: Comparison between *os-ov* (implemented TLD) and *ps-pv* (our proposal) in TLD dataset. Cells on bold represent the highest f-measure.

We have also tested both variations in ALOV300++ dataset. F-scores by class and variation are shown in Figure 5.11. We can see balanced results in this dataset. *os-ov* outperformed *ps-pv* by wider margins in classes as Specularity, but *ps-pv* outperformed *os-ov* in more aspects. Some results are consistent with the results achieved with TLD. For instance, our method achieved better score in Shape class. When comparing overall perfomance by survival curves, the balance is clear (Fig. 5.12). It is important to highlight that the sequence order in these curves may be different. They did not achieve high score for the same sequence 1, for example, they achieved high score for the first sequence in terms of each overall performance.

Concerning the computational cost, our method achieved 5.75 *fps* against 7.15*fps* from *os-ov* in TLD dataset. We believe that the frame rate decreased because more samples are inserted in the detector training set, affecting the classification cost. But it was a small decrease.

5.3.5 TLD - PUBLISHED RESULTS

As said in the beginning of this section, we have chosen to fully implement the TLD tracker to conduct our experiments instead of using the published results. In this subsection, we compare the results from both the implemented *os-ov* and the original TLD paper results (Tab. 5.5). In addition, we also show the results that we achieved with the demonstration version made available by the authors.

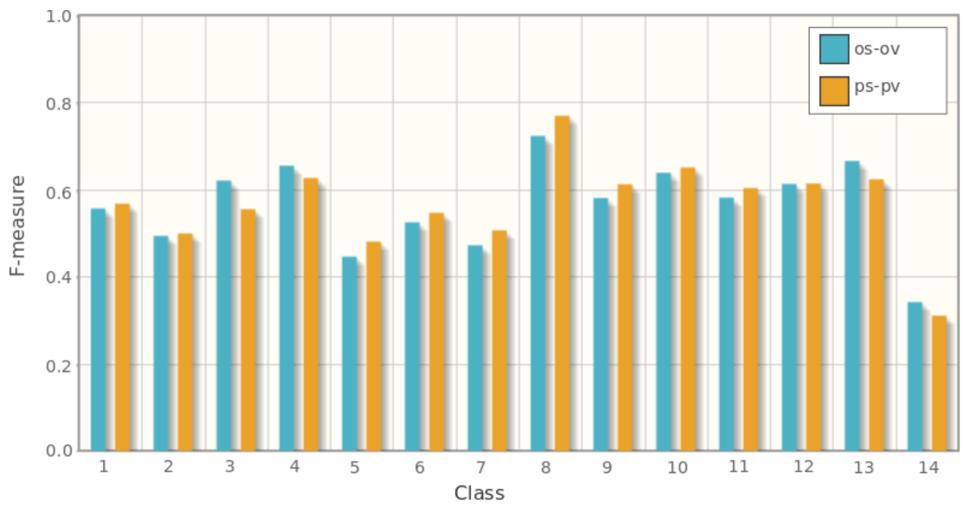


Figure 5.11: Comparison between *os-ov* (implemented TLD) and *ps-pv* (our proposal) in ALOV300++ dataset using F-measure. The classes are 1-Light, 2-SurfaceCover, 3-Specularity, 4-Transparency, 5-Shape, 6-MotionSmoothness, 7-MotionCoherence, 8-Clutter, 9-Confusion, 10-LowConstrast, 11-Occlusion, 12-MovingCamera, 13-ZoomingCamera and 14-LongDuration.

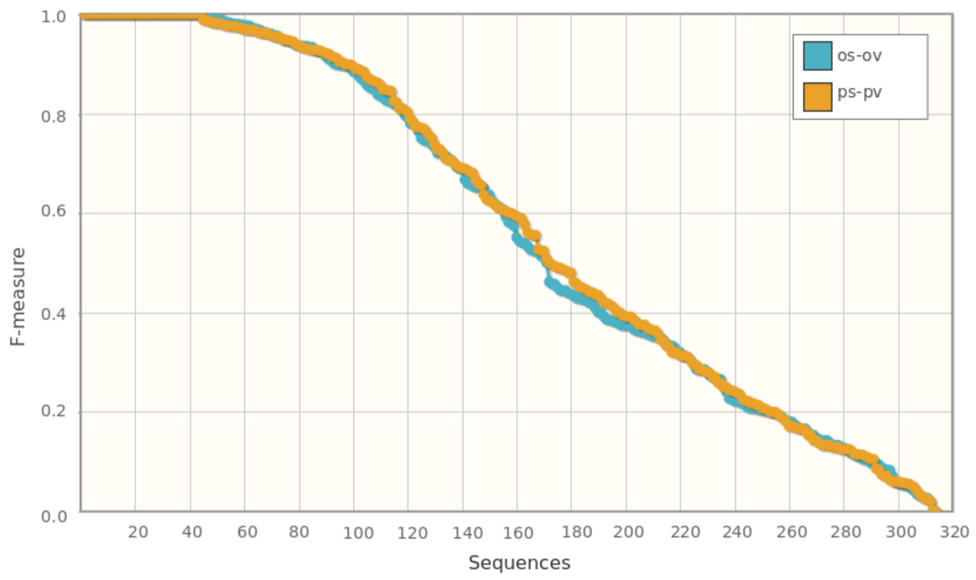


Figure 5.12: Survival curves for both *os-ov* and *ps-pv* variations in ALOV300++ dataset using f-measure.

Sequence	Frames	*os-ov* P / R / F	Demo P / R / F	TLD P / R / F
David	761	**1.00 / 1.00 / 1.00**	0.99 / 0.99 / 0.99	**1.00 / 1.00 / 1.00**
Jumping	313	1.00 / 0.98 / 0.99	**1.00 / 1.00 / 1.00**	**1.00 / 1.00 / 1.00**
Pedestrian 1	140	0.39 / 0.13 / 0.19	**1.00 / 1.00 / 1.00**	**1.00 / 1.00 / 1.00**
Pedestrian 2	338	0.99 / 0.68 / 0.81	0.24 / 0.30 / 0.27	**0.89 / 0.92 / 0.91**
Pedestrian 3	184	**0.99 / 1.00 / 0.99**	**0.99 / 1.00 / 0.99**	**0.99 / 1.00 / 0.99**
Car	945	0.87 / 0.81 / 0.84	**0.93 / 0.99 / 0.96**	0.92 / 0.97 / 0.94
Motocross	2665	0.78 / 0.58 / 0.67	**0.90 / 0.92 / 0.91**	0.89 / 0.77 / 0.83
Volkswagen	8576	**0.88 / 0.92 / 0.90**	0.69 / 0.91 / 0.79	0.80 / 0.96 / 0.87
Carchase	9928	0.96 / 0.21 / 0.35	0.83 / 0.23 / 0.36	**0.86 / 0.70 / 0.77**
Panda	3000	**0.64 / 0.70 / 0.67**	0.42 / 0.46 / 0.44	0.58 / 0.63 / 0.60
mean	26850	0.88 / 0.59 / 0.64	0.75 / 0.61 / 0.61	**0.82 / 0.81 / 0.81**

Table 5.5: Comparison between implemented TLD (*os-ov*), demonstration version (Demo) and published results (TLD) in TLD dataset. Bold cells represent the highest f-measure.

TLD tracker contains a large set of parameters and some of them are not reported in the paper. In our available time, we exploited extensively the paramater space of *os-ov* to achieve exactly the same result of the paper, without success. We were not able to achieve TLD results with the demonstration version either. So, we believe that the published results may be achieved with different initial positions and/or parameter settings. To contour this reproducibility problem or the lack of information of the original paper, we have chosen to compare the original and proposed *selector/validator* in our implementation. Our method, however, reached some of the published results and even outperformed others.

Sequence	Frames	TLD P / R / F	*ps-pv* P / R / F
David	761	**1.00 / 1.00 / 1.00**	**1.00 / 1.00 / 1.00**
Jumping	313	**1.00 / 1.00 / 1.00**	**1.00 / 1.00 / 1.00**
Pedestrian 1	140	**1.00 / 1.00 / 1.00**	0.37 / 0.21 / 0.26
Pedestrian 2	338	0.89 / 0.92 / 0.91	**0.90 / 0.97 / 0.93**
Pedestrian 3	184	**0.99 / 1.00 / 0.99**	0.96 / 1.00 / 0.98
Car	945	0.92 / 0.97 / 0.94	**0.93 / 0.99 / 0.96**
Motocross	2665	0.89 / 0.77 / 0.83	**0.88 / 0.86 / 0.87**
Volkswagen	8576	0.80 / 0.96 / 0.87	**0.93 / 0.91 / 0.92**
Carchase	9928	**0.86 / 0.70 / 0.77**	0.95 / 0.51 / 0.67
Panda	3000	0.58 / 0.63 / 0.60	**0.80 / 0.87 / 0.83**
mean	26850	**0.82 / 0.81 / 0.81**	**0.92 / 0.76 / 0.81**

Table 5.6: Comparison between published results (TLD) and our proposal *ps-pv* in TLD dataset. Cells on bold represent the highest f-measure.

Comparing the published results with our proposal *ps-pv* (Tab. 5.6), we can see that both achieved the same f-measure. We achieved a higher precision against a higher recall from TLD. Our worst score was on Pedestrian1 which presents considerable camera motion. This was a challenging sequence for all of our variations. On the other hand, we continue achieving good results on Panda sequence that presents significant object deformation.

5.3.6 LITERATURE COMPARISONS

We also compare our method with the state-of-the-art in each dataset. In TLD dataset, we compare our results with the ALIEN tracker (PERNICI; BIMBO, 2013). The results published in ALIEN paper consider the overlap threshold equal to 50% and do not consider the Panda sequence. We evaluate our and published TLD outputs under the same conditions. These results are shown in Table 5.7.

Video	Frames	TLD *P / R / F*	*ps-pv* (our) *P / R / F*	*ALIEN* *P / R / F*
David	761	**1.00 / 1.00 / 1.00**	**1.00 / 1.00 / 1.00**	0.99 / 0.98 / 0.99
Jump	313	**0.99 / 0.99 / 0.99**	0.91 / 0.91 / 0.91	0.99 / 0.87 / 0.92
Ped1	140	**1.00 / 1.00 / 1.00**	0.28 / 0.16 / 0.20	**1.00 / 1.00 / 1.00**
Ped2	338	0.89 / 0.92 / 0.91	0.86 / 0.92 / 0.89	**0.93 / 0.92 / 0.93**
Ped3	184	**0.99 / 1.00 / 0.99**	0.96 / 1.00 / 0.98	1.00 / 0.90 / 0.95
Car	945	0.92 / 0.97 / 0.94	0.93 / 0.99 / 0.96	**0.95 / 1.00 / 0.98**
Moto	2665	0.67 / 0.58 / 0.62	0.62 / 0.61 / 0.62	**0.69 / 0.81 / 0.74**
Volks	8576	0.54 / 0.64 / 0.59	0.54 / 0.53 / 0.54	**0.98 / 0.89 / 0.93**
Chase	9928	0.50 / 0.40 / 0.45	0.82 / 0.44 / 0.57	**0.73 / 0.69 / 0.70**
Mean	26850	0.58 / 0.57 / 0.58	0.71 / 0.55 / 0.60	**0.84 / 0.80 / 0.82**

Table 5.7: Comparison between *ps-pv* (our proposal) and ALIEN tracker (state-of-the-art). Cells on bold represent the highest f-measure.

We can see that by using a more rigorous hit criterion for bounding boxes, we achieved better results than TLD. This means a better fitting of our estimated trajectories with the expected ones. Considering ALIEN tracker, we achieved slightly better results in David an Pedestrian3 sequences and comparable performance in many sequences. But ALIEN tracker outperforms both, showing that estimating object position by keypoints may be more effective than by subimages.

In ALOV300++, we compare our method with the state-of-the-art Struck tracker, proposed by Hare et al. (2011) (Fig. 5.13). We can see that our method achieved better results in Shape and ZoomingCamera, reinforcing our point about Shape class. However, the overall performance of Struck also outperformed our results.

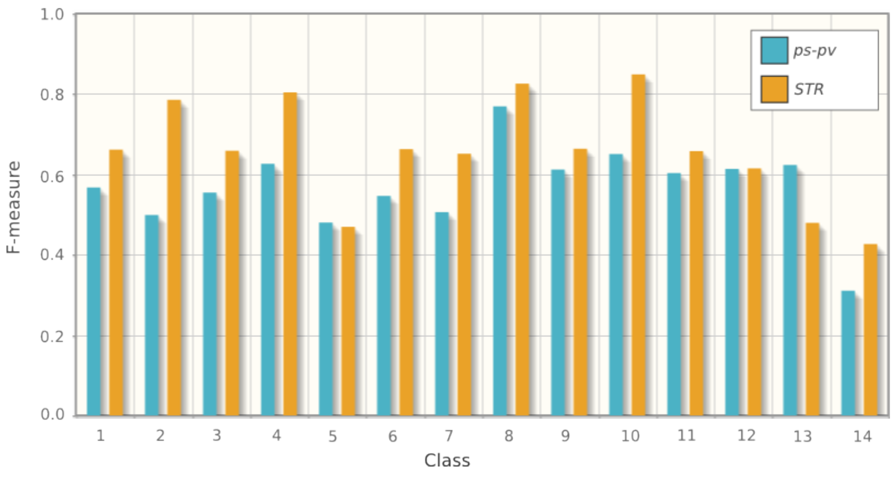


Figure 5.13: Comparison between our proposed method *ps-pv* and the state-of-the-art Struck. The classes are 1-Light, 2-SurfaceCover, 3-Specularity, 4-Transparency, 5-Shape, 6-MotionSmoothness, 7-MotionCoherence, 8-Clutter, 9-Confusion, 10-LowConstrast, 11-Occlusion, 12-MovingCamera, 13-ZoomingCamera and 14-LongDuration.

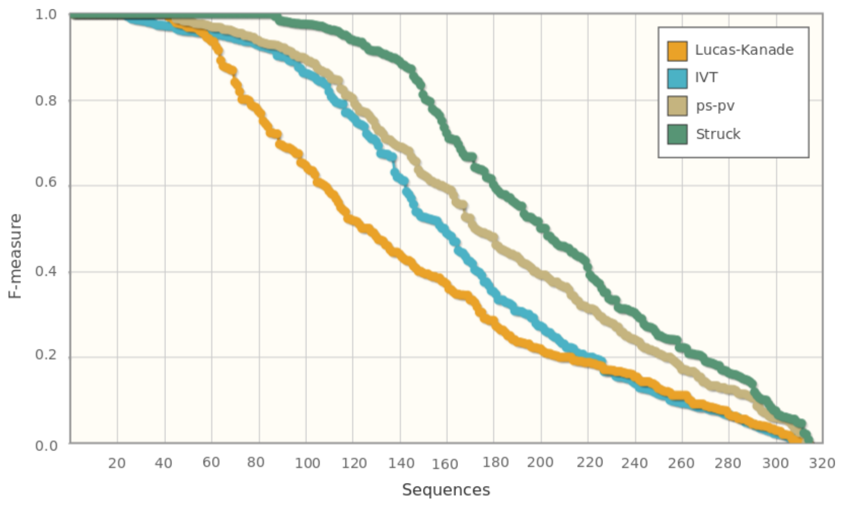


Figure 5.14: Survival curves for both methods in ALOV300++ dataset using f-measure.

In Figure 5.14, we compare the overall performance of our method with other trackers in literature: the motion trackers Lucas-kanade by Baker and Matthews (2004) and IVT by Ross et al. (2008) and the discriminative tracker Struck by (HARE et al., 2011). Our method outperforms Lucas-Kanade and IVT, but Struck achieved higher scores.

6 CONCLUSION

In this work, we proposed a general framework for object tracking combining multiple trackers. The central component of our framework is the *mediator*, the component responsible for integrating the outcomes and updating the trackers. We proposed a basic premise for any mediator: it should make decisions using their own knowledge and strategy, i.e., work independently of its mediated trackers. The *mediator* is composed of a *selector* that chooses the final response and many *validators* that update the trackers. Following this framework, we presented a novel method for object tracking. Our method includes a *mediator* and two trackers from TLD: the motion tracker and the detector. The *mediator* contains one *selector* and one *validator* that updates the detector using motion tracker responses, both independently defined. In TLD, selection and validation methods, although with other names, were based solely on the detector strategy and model.

We have built our *selector* based on invariant features from the first appearance. This was a good option because it works only with genuine information. However, we showed that this makes our *selector* strongly dependent on the quality of the first appearance, as seen in the similarity value decrease throughout the David ground truth.

We presented quantitative results of our methods in comparison with the original TLD. Our results show that replacing one non-independent component at a time is not effective. But replacing both outperforms the original method. We also achieved lower scores when using *selector* and *validator* based on the same metric and model. So, we argue that every component must be independent from the others since the failures in one may spread to the whole system.

Our method can deal with objects that present high appearance variations as orientation changes and deformations. This was observed in both tested datasets, in comparison with TLD and other methods from the literature. However, under dramatic camera motion, our method fails. When comparing with other methods, our method outperform some of them, but does not outperform the state-of-the-art.

In our experiments, we noticed some drawbacks of our method. The *validator* is vulnerable to the tracker re-initializations, since it uses the context of the tracker responses. Future works may include improvements in the *validator* in order to recover to older and more reliable contexts. The *selector* might be improved by learning a distance metric from the few labeled samples for each aspect. Instead of collecting new samples for the *selector*, we could adjust the features weight during the tracking, according to the appearance changes. Since the *selector* is a classifier working with a reduced number of candidates, if compared with the detector, we could explore larger descriptors using other aspects from the appearance.

REFERENCES

BAKER, S.; MATTHEWS, I. Lucas-kanade 20 years on: A unifying framework. **International journal of computer vision**, Springer, v. 56, n. 3, p. 221–255, 2004.

BOUGUET, J.-Y. Pyramidal implementation of the lucas lanade feature tracker description of the algorithm. **Intel Microprocessor Research Labs**, v. 5, 2001.

COMPORT, A. I.; MARCHAND, E.; PRESSIGOUT, M.; CHAUMETTE, F. Real-time markerless tracking for augmented reality: the virtual visual servoing framework. **Visualization and Computer Graphics, IEEE Transactions on**, IEEE, v. 12, n. 4, p. 615–628, 2006.

DALAL, N.; TRIGGS, B. Histograms of oriented gradients for human detection. In: IEEE. **Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on**, 2005. v. 1, p. 886–893.

HANSEN, L. K.; SALAMON, P. Neural network ensembles. **IEEE transactions on pattern analysis and machine intelligence**, IEEE Computer Society, v. 12, n. 10, p. 993–1001, 1990.

HARE, S.; SAFFARI, A.; TORR, P. H. Struck: Structured output tracking with kernels. In: IEEE. **Computer Vision (ICCV), 2011 IEEE International Conference on**, 2011. p. 263–270.

KALAL, Z.; MATAS, J.; MIKOLAJCZYK, K. P-N learning: Bootstrapping binary classifiers by structural constraints. **Conference on Computer Vision and Pattern Recognition**, 2010.

KALAL, Z.; MIKOLAJCZYK, K.; MATAS, J. Forward-backward error: Automatic detection of tracking failures. In: IEEE. **International Conference on Pattern Recognition (ICPR)**, 2010. p. 23–26.

KALAL, Z.; MIKOLAJCZYK, K.; MATAS, J. Tracking-learning-detection. In: , 2012. v. 34, n. 7.

LEE, D.-Y.; SIM, J.-Y.; KIM, C.-S. Multihypothesis trajectory analysis for robust visual tracking. In: **Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition**, 2015. p. 5088–5096.

LOWE, D. G. Distinctive image features from scale-invariant keypoints. **International journal of computer vision**, Springer, v. 60, n. 2, p. 91–110, 2004.

LUCAS, B. D.; KANADE, T. An iterative image registration technique with an application to stereo vision. In: **IJCAI**, 1981. v. 81, p. 674–679.

MATTHEWS, I.; ISHIKAWA, T.; BAKER, S. The template update problem. **IEEE Transactions on Pattern Analysis & Machine Intelligence**, IEEE, n. 6, p. 810–815, 2004.

OJALA, T.; PIETIKÄINEN, M.; HARWOOD, D. A comparative study of texture measures with classification based on featured distributions. **Pattern recognition**, Elsevier, v. 29, n. 1, p. 51–59, 1996.

OZUYSAL, M.; FUA, P.; LEPETIT, V. Fast keypoint recognition in ten lines of code. In: IEEE. **Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on**, 2007. p. 1–8.

PÉREZ, P.; HUE, C.; VERMAAK, J.; GANGNET, M. Color-based probabilistic tracking. In: **Computer vision - ECCV 2002**, 2002. p. 661–675.

PERNICI, F.; BIMBO, A. D. Object tracking by oversampling local features. In: , 2013.

ROSENBERG, C.; HEBERT, M.; SCHNEIDERMAN, H. Semi-supervised self-training of object detection models. In: , 2005.

ROSS, D. A.; LIM, J.; LIN, R.-S.; YANG, M.-H. Incremental learning for robust visual tracking. **International Journal of Computer Vision**, Springer, v. 77, n. 1-3, p. 125–141, 2008.

SCHWARTZ, W. R.; DAVIS, L. S. Learning discriminative appearance-based models using partial least squares. In: IEEE. **Computer Graphics and Image Processing (SIBGRAPI), 2009 XXII Brazilian Symposium on**, 2009. p. 322–329.

SHAH, M.; JAVED, O.; SHAFIQUE, K. Automated visual surveillance in realistic scenarios. **IEEE MultiMedia**, IEEE, n. 1, p. 30–39, 2007.

SHI, J.; TOMASI, C. Good features to track. In: IEEE. **Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on**, 1994. p. 593–600.

SMEULDERS, A. W.; CHU, D. M.; CUCCHIARA, R.; CALDERARA, S.; DEHGHAN, A.; SHAH, M. Visual tracking: An experimental survey. **Pattern Analysis and Machine Intelligence, IEEE Transactions on**, IEEE, v. 36, n. 7, p. 1442–1468, 2014.

VIOLA, P.; JONES, M. Rapid object detection using a boosted cascade of simple features. In: IEEE. **Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on**, 2001. v. 1, p. I–511.

ZHANG, T.; LIU, S.; XU, C.; YAN, S.; GHANEM, B.; AHUJA, N.; YANG, M.-H. Structural sparse tracking. In: **Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition**, 2015. p. 150–158.