Universidade Federal de Juiz de Fora
Instituto de Ciências Exatas
Bacharelado em Ciência da Computação

Study and implementation of an object tracker in videos

Helena de Almeida Maia

JUIZ DE FORA
DEZEMBRO, 2014

Study and implementation of an object tracker in videos

Helena de Almeida Maia

Universidade Federal de Juiz de Fora

Instituto de Ciências Exatas

Departamento de Ciência da Computação

Bacharelado em Ciência da Computação

Orientador: Marcelo Bernardes Vieira

JUIZ DE FORA

DEZEMBRO, 2014

Study and implementation of an object tracker in videos

Helena de Almeida Maia

MONOGRAFIA SUBMETIDA AO CORPO DOCENTE DO INSTITUTO DE CIÊNCIAS EXATAS DA UNIVERSIDADE FEDERAL DE JUIZ DE FORA, COMO PARTE INTEGRANTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE BACHAREL EM CIÊNCIA DA COMPUTAÇÃO.

Aprovada por:

Marcelo Bernardes Vieira
Doutor em Ciência da Computação

Marcelo Caniato Renhe
Mestre em Engenharia de Sistemas e Computação

Raul Fonseca Neto
Doutor em Engenharia de Sistemas e Computação

JUIZ DE FORA
11 DE DEZEMBRO, 2014

Aos meus pais, Mauricio e Fátima

Às minhas irmãs, Carol, Mariana e Laura

Ao meu namorado, Gabriel

Aos meus amigos

Resumo

Rastreamento em vídeos fornece a trajetória de um objeto no tempo. Diversas pesquisas em visão computacional e realidade aumentada dependem desta tarefa. Existem várias soluções propostas na literatura e a maior parte delas enfrentam desafios relacionados ao rastreamento a longo prazo. Neste cenário, rastreadores costumam falhar, exigindo uma reinicialização. Para evitar estas falhas, várias pesquisas foram feitas com o intuito de combinar rastreadores e detectores em tempo de execução. O presente trabalho tem como objetivo o estudo de um rastreador a longo prazo em vídeo, bem como sua implementação.

Palavras-chave: rastreamento a longo prazo, fluxo mediano, detector estatístico, rastreamento-aprendizado-detecção.

Abstract

Tracking in videos provides the object trajectory over time. Several researches on computer vision and augmented reality rely on this task. There is a large variety of trackers proposed on literature. Most of these approaches face challenges related to long-term tracking. In this scenario, trackers tend to fail requiring restarting. To avoid these failures, over the past years, we have seen a growing interest on combining trackers and detectors at runtime. The present work intends to present a study of a long-term tracker in video as well as its implementation.

Keywords: long-term tracking, median flow, statistical detector, tracking-learning-detection.

Agradecimentos

Agradeço aos meus pais pelo amor e pelo constante incentivo aos estudos e às minhas irmãs por todo o apoio. Ao meu namorado, Gabriel, companheiro e paciente, por me motivar a persistir sempre.

Aos meus amigos da UFJF, Bruno, Carol, Luiz, Ronaro, Sandra, Tay e Weslley que estiveram comigo durante toda a graduação tanto nas comemorações quanto nos momentos difíceis. Agradeço à todos os amigos do Grupo de Computação Gráfica que auxiliaram com revisões do texto e com discussões sobre o assunto e àqueles que foram rastreados involuntariamente. Agradeço, especialmente, à equipe de visão computacional da qual eu fiz parte, Ana, Fábio e Virgínia e aos amigos Caetano e Luciano pela atenção e disponibilidade em ajudar.

Aos professores do DCC que contribuíram para a minha formação. Em especial, ao professor Marcelo Bernardes pela oportunidade e por suas meticulosas correções até o último minuto e ao professor Rodrigo Luis por seus ensinamentos.

Agradeço a todos que direta ou indiretamente fizeram parte da minha formação.

"But it was all right, everything was all right, the struggle was finished. He had won the victory over himself. He loved Big Brother."

George Orwell (1984)

Contents

List of Figures

List of Tables

List of abbreviations

2D	Two-dimensional
3D	Three-dimensional
FB	Forward-Backward Error
NCC	Normalized Cross-Correlation
SSD	Sum of Squared Differences
TLD	Tracking-learning-detection

1 Introduction

Object tracking is a fundamental task for several areas of research such as surveillance and augmented reality. Tracking provides the object position over time, so the system will be able to analyze object behavior or to produce new objects over it.

There is a large variety of trackers on literature differing mainly on the object model, supported transformations and motion estimation. The reason is that different applications and environments require different abilities from the tracker. Some approaches consist in building 3D models of rigid objects offline and estimating the affine transformation parameters on each frame (Oikawa et al., 2012). In these approaches, the object geometry must be known a priori and they usually cannot handle deformations. For surveillance purposes, we need a more flexible tracker. There are several approaches based on template matching (Kalal et al., 2010; Pernici and Del Bimbo, 2013). Template is an object representation, for instance a color histogram or a set of key points and the tracker searches for the patch which minimizes an error function. Generally, this search is made between pairs of consecutive frames and the object found is used for the next search. Many of these object trackers are based on point trackers like pyramidal Lucas-Kanade tracker (Bouguet, 2001) and the object is defined by a bounding box.

There are some classical challenges on object tracking. A common challenge is the change in the object appearance caused by illumination variations, object occlusions, among others. Trackers also tend to accumulate errors and to fail after some time. These issues are even worse when the tracking is made in a large amount of time, the so-called long-term tracking.

Over the past years, we have seen a growing interest on combine trackers and detectors, at runtime, to address these issues. Detector requires a training stage to model the target object and, consequently, it is more capable to recognize it. It checks the whole frame for the known template and is capable to detect when the object is out of the scene. However, it may be confused with similar objects, since it does not consider previous position, besides requiring a large dataset to learn object appearance. Trackers

are capable to adapt to new object appearances, but they fail in recognize the object more frequently.

A notable approach was proposed by Kalal et al. (2012). It combines object tracking, detection and a semi-supervised learning component in a framework called TLD or, as it is known, the predator. With the learning component, the system is able to update the object model and retrain the detector.

In this work, we implement a long-term tracker which is similar to TLD. It is composed of a tracker and a detector without the learning component. Thus, we evaluate the ability of the detector to correct tracker responses without retraining in a long-term scenario.

1.1 Problem definition

Given a bounding box defining an object, we want to find its trajectory throughout a long-term video. This implies a frame-to-frame tracking which can handle illumination variations and object occlusions in addition to detect if the object disappears of the scene. Thus, given an object in time t, which is actually an appearance of the real object, we want to estimate the new position in time $t + 1$, its scale factor, or to indicate that the object is out of the scene.

1.2 Objectives

The main objectives of this work are the study and implementation of a tracker coupled with a detector to avoid tracking failures. We analyze the limitations of this combination in a long-term scenario. The tracker accuracy is evaluated using TLD dataset proposed by Kalal et al. (2010). We also present a study of each component individually considering some classical challenges on object tracking. Finally, we present and discuss some qualitative results.

2 Fundamentals

In this chapter, we present some fundamental concepts for tracking and detection tasks. But, first of all, we introduce some general definitions.

Definition 2.0.1. (Image) An image is a 2D continuous signal $I : \mathbb{R}^2 \rightarrow \mathbb{R}^c$ which associates a point $\vec{q} = (x, y)$ to a brightness vector $\vec{v} \in \mathbb{R}^c$. Each vector element represents the channel intensity for a point. A digital image I is a discrete representation of an image that usually is 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 $[0, 1]$. By convention, the image origin is located at the upper left point.

Definition 2.0.2. (Patch) A patch is a restriction of the image domain. It can be seen as an $m' \times n'$ sub-image where $m' < m$ and $n' < n$. Let $\vec{q_1} = (x_1, y_1)$ be the initial point of the patch in an image I. The patch P is defined as:

$$P(x, y) = I(x_1 + x, y_1 + y),$$

where $0 \leq x < m'$ and $0 \leq y < n'$. The terminal point of the patch in I is $\vec{q_2} = (x_1 + n', y_1 + m') = (x_2, y_2)$.

Definition 2.0.3. (Bounding box) A bounding box is a minimum rectangle which contains the object. Formally, it is given by its minimum and maximum coordinates $B = \{x_{min}, y_{min}, x_{max}, y_{max}\}$. To indicate that the object is out of the scene, we set an invalid value for coordinates.

Definition 2.0.4. (Video) A video V is a sequence of images or frames ordered by time, i.e., $V = \{I_1, \cdots, I_k\}$. For tracking purposes, we consider smooth sequences without sudden cuts of scene. That is, we only accept progressive changes, including illumination variations and object occlusions.

Definition 2.0.5. (Integral image) Integral image was proposed by Viola and Jones

(2001) and its elements $I_{int}(x, y)$ are given by:

$$I_{int}(x, y) = \sum_{0 \leq x' \leq x} \sum_{0 \leq y' \leq y} f(I(x', y')),$$

where I is an image, f a function over I values, $x < n$ and $y < m$.

Let S be an image defined as

$$S(x, y) = \sum_{0 \leq y' \leq y} f(I(x', y')),$$

i.e., an image where each element contain the column sum up to y. Considering $S(x, 0) = f(I(x, 0))$ and $I_{int}(0, y) = S(0, y)$ for all x and y, the integral image can be calculated as:

$$\begin{cases} S(x, y) = S(x, y - 1) + f(I(x, y)) \\ I_{int}(x, y) = I_{int}(x - 1, y) + S(x, y) \end{cases}$$

Now, suppose we want to find the integral value for a patch P within I defined by the points $\vec{q_1} = (x_1, y_1)$ and $\vec{q_2} = (x_2, y_2)$:

$$I_{patch}(P) = \sum_{x_1 \leq x' \leq x_2} \sum_{y_1 \leq y' \leq y_2} f(I(x', y')).$$

As depicted in Figure 2.1, $I_{patch}(P)$ can be calculated with only four accesses to the integral image:

$$I_{patch}(P) = I_{int}(x_2, y_2) - I_{int}(x_2, y_1) - I_{int}(x_1, y_2) + I_{int}(x_1, y_1).$$

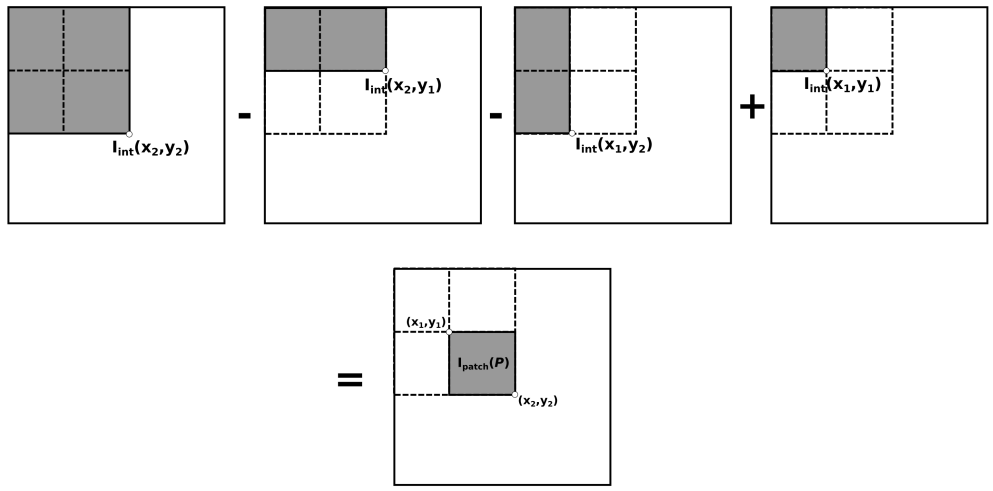

Figure 2.1: Integral value for a patch P.

2.1 Tracker concepts

As mentioned before, recent approaches combine tracking and detection tasks at runtime. These combinations are also called trackers because they share the same goal. However, most of the earliest works were based on a pure tracker.

Given an object position in a frame, the task of a pure tracker is to find the position in a second frame. In other words, the tracker must estimate an affine transformation which will give the second location of the object. Trackers assume that the time difference between the two frames is small and, thus, the second position is close to the first one. For this reason, the search can be made on a small area around the initial position. This task may also be extended for multiple frames and each object found become the reference for the next search.

2.1.1 Object

The object definition itself has a broad meaning. It may be a person, a car or even a point. Each class of object requires different supported transformations. Point trackers, for instance, usually use point displacements to describe movement while rigid objects motion may be represented by translation, rotation and scale of the original appearance.

The input object is, in fact, a view of the real object. During the video, we can

have a completely different appearance of the same object. This scenario is even worse with deformable objects. Trackers are usually capable of dealing with gradual changes and learning new appearances.

There are many ways to represent the object: by histograms, articulated models, 3D meshes, among others. The most common is by feature points. In this work, we consider rigid and deformable objects given by the user, i.e., selected bounding box which marks the object, and they are going to be represented by feature points.

2.1.2 Trajectory

Tracking on a sequence of frames will give us a sequence of object states. In some works, mainly 3D approaches, the object state consists of a parameter set containing, for instance, the rotation angle related to the initial appearance. This state is known as the object pose. For most of 2D approaches, the object state is a pair of points, defining a bounding box, which represents the location and extension of the object. A sequence of object states is called trajectory. Formally, the trajectory is given by $T = \{B_i\}$ where B_i is a bounding box, $i \in \{1, \cdots, k\}$ and k is the number of frames or the trajectory size. The aspect ratio of the initial bounding box is kept through the whole trajectory. Note that this first bounding box must be given. It can be obtained from a detector, from the user or, in a controlled environment, by background subtraction.

Trackers always output smooth trajectories. This means that it will estimate a transformation for every frame, even if the object moves out of the scene or is occluded, and the estimated position is close to the previous. For trackers, similarity is not more important than proximity. It will not select an identical object in the scene if it is out of the search area, and might retrieve another object if it is the most similar in the neighborhood. Therefore, trackers assume that the object is always visible and cannot handle sudden and large movements. If the video does not satisfy these conditions, the tracker tends to fail.

2.1.3 The drift problem

As we said, trackers are capable of adapting to new object appearances and dealing with progressive changes. Since there is no guarantee that every point in the new appearance belong to the original object, the tracker can follow a wrong object or get stuck on a cluttered background and never recover the previous target. This problem is known as the drift problem, i.e., the error accumulation during the track. For the best of our knowledge, no pure tracker can recover from the drift problem. That is why several works include a detector component. Figure 2.2 shows an example of drift failure in sequence car from TLD dataset (Kalal et al., 2010). The car is occluded by a tree in the second image and the tracker cannot recover the car again.

Figure 2.2: Example of drift failure in sequence car from TLD dataset (Kalal et al., 2010).

2.2 Detector concepts

The main task of a detector is to decide whether the object is in the scene or not. If the object is visible, the detector must find out its location. Generally, detector focuses on learning a class pattern instead of discriminate interclass objects (Amit, 2002). For this reason, similar objects in the scene may confuse the detector. The detector task can be seen as a classification problem: given a sample set, the detector classifies each candidate patch as positive (object) or negative (background and other objects). To make this decision, detector must learn object appearances and build a model from them.

According to Ozuysal et al. (2007), there are two main groups of approaches for detection: based on invariant descriptor and on statistical models. Basically, detectors based on invariant descriptor (Lowe, 1999; Bay et al., 2006) focuses on extracting features invariant to affine transformations and, at runtime, matching the descriptors. This way,

the detector will be able to deal with different appearances of the object. Statistical models (Lepetit and Fua, 2006; Ozuysal et al., 2007) are based on a training set (authentic or synthetic) for object modeling and a classifier to learn object pattern. Here, we focus on statistical model. Since we are working with 2D views of a 3D object, the training set must include variations of object perspective. For instance, Lepetit et al. (2004) synthesize views of the original object using affine transformations.

2.2.1 Binary test

The aforementioned statistical models extract feature points from the object for matching. At runtime, a point of a candidate object must be classified as a known feature point or not. The recognition is made through intensity comparisons in the feature and in the candidate surrounding patch. Ozuysal et al. (2007) proposes a set of binary tests $F = \{f_1(\vec{q}), \cdots, f_b(\vec{q})\}$ where $f_i(\vec{q})$ is a function given by:

$$f_i(\vec{q}) = \begin{cases} 1, & \text{if } I(\vec{r}_{0,i}) < I(\vec{r}_{1,i}) \\ 0, & otherwise \end{cases}.$$

Points $\vec{r}_{0,i}$ and $\vec{r}_{1,i}$ are randomly chosen offline and belong to the surrounding patch of $\vec{q}$. The binary tests in a point neighborhood give us a code which is the concatenation of the results. This code is used for point matching. An example of code generation is shown in Figure 2.3 for two binary tests where $r_{0,0} = (0, 0)$, $r_{1,0} = (0, 1)$, $r_{0,1} = (2, 0)$ and $r_{1,1} = (2, 1)$. The code generated for the feature patch was 01 while for candidate patch was 00.

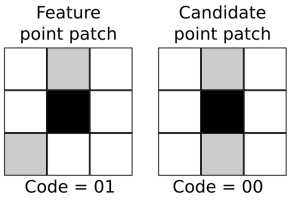

Figure 2.3: Example of codes for a feature and a candidate point.

By creating new views of the object, we also have new views of a feature point.

These corresponding points define a class. Thus, a candidate point may be similar to any view of a feature point. Ozuysal formulates this problem in a naive Bayesian classification framework. So, we calculate the posterior probability for a point to be classified as c_j, given that it has the code $f_{code}(\vec{q})$, i.e., $P(C = c_j \mid f_{code}(\vec{q}))$. C is a random variable that represents a class and can take on a feature index or -1 to indicate an unknown point. Note that for b binary tests, we need to store 2^b posteriors values for each class. For this reason, Ozuysal proposes to use an ensemble of Bayesian classifiers.

2.2.2 Ensemble classifier

An ensemble is a set of base classifiers whose individual votes are combined into a majority decision for a new sample. If each classifier makes independent errors and has at least 50% accuracy, then the collective response outperforms individual decisions (Hansen and Salamon, 1990). To address the independence criteria, Ozuysal et al. (2007) proposes the fern structure. Ferns are equal-sized partitions of the F set. Each ensemble member is associated with a different fern set and thus, tends to make independent errors. Moreover, this approach reduces the computational effort to store the posteriors. Each fern produces $2^{\frac{b}{l}}$ posteriors for each class and $l \cdot 2^{\frac{b}{l}}$ for the whole set of ferns, where l is the number of ferns. Formally, a fern F_i is given by:

$$F_i = \{f_{\sigma(i,1)}(\vec{q}), \cdots, f_{\sigma(i,t)}(\vec{q})\},$$

where $\sigma(i, j)$ is a random permutation of F indexes and $t = \frac{b}{l}$. The global response of the ensemble is given by:

$$P(C = c_j \mid f_{code}(\vec{q})) = \prod_{i=1}^{l} P(C = c_j \mid f^i_{code}(\vec{q})).$$

Finally, for a given point $\vec{q}$, the highest probability corresponds to the most similar class, i.e.:

$$c = \max_{c_j} P(C = c_j \mid f_{code}(\vec{q})).$$

The posterior probability and the random permutation are calculated offline once. Ozuysal shows that the posterior probability is approximately:

$$P(C = c_j \mid f^i_{code}(\vec{q})) \approx \frac{n_{i,j} + 1}{\sum_{a=1}^{l} (n_{a,j} + 1)},$$

where $n_{i,j}$ is the number of training samples that generate the code f^i_{code} and has the label c_j.

3 Long-term tracker

In this chapter, we present the long-term tracker. It was written in C++ language with the OpenCV (Open source Computer Vision) library. Sections 3.1 and 3.2 contains individual explanation and implementation of the tracker and detector, respectively. Both output a bounding box for each frame of the video. In the last section, we show the combination of tracker and detector responses in a unified component.

3.1 Tracker

In this section, we detail tracker implementation proposed in Forward-Backward Error (Kalal et al., 2010) and Tracking-Learning-Detection (Kalal et al., 2012). The tracker receives as input a video and a bounding box defining the object in the first frame. The main algorithm is presented below.

Algorithm 1: Median Flow Tracker.

Data: Video $V = \{I_1, \cdots, I_k\}$, Bounding Box B_1.
Result: Trajectory $T = \{B_1, \cdots, B_k\}$.
begin
  foreach $I_i \in V - \{I_k\}$ **do**
    $G \leftarrow$ buildGrid(I_i, B_i);
    $G' \leftarrow$ pyramidalTracker(I_i,I_{i+1},G);
    $(G, G') \leftarrow$ filterPoints(G,G');
    $B_{i+1} \leftarrow$ medianFlow(G,G',B_i);
  end foreach
end

For each frame, the tracker selects equally spaced points within the bounding box in the function *buildGrid()*. It returns a 10×10 grid and these points are used to estimate object motion. The other functions are explained in the following subsections: *pyramidalTracker()* in Subsection 3.1.1, *filterPoints()* in Subsection 3.1.2 and Subsection 3.1.3, and *medianFlow()* in Subsection 3.1.4. For each frame, a bounding box is produced containing the object for the next frame. The output of the method is the set of bounding boxes, i.e., the object trajectory.

3.1.1 Pyramidal Lucas-Kanade tracker

Lucas and Kanade (1981) proposed a method to compute optical flow. Let $\vec{q} = (x, y)$ be a point in the first image I and J a second image. We want to find the displacement vector $\vec{d} = (d_x, d_y)$ such that the neighborhood of $\vec{q}$ in I is similar to a neighborhood of $\vec{q}+\vec{d}$ in J. Each neighborhood is defined as a $(2w+1) \times (2w+1)$ patch. The error measure between two points is given by the L_2 norm. Thus, for the whole patch, we should minimize the error function given by:

$$e(\vec{d}) = \sum_{(x',y')} [I(x', y') - J(x' + d_x, y' + d_y)]^2,$$

where $x - w \leq x' \leq x + w$ and $y - w \leq y' \leq y + w$. This method assumes small and uniform displacement within the patch. Lower w values keep details while higher values are robust to noises and illumination changes and allows large movements.

For keeping the accuracy and robustness, Bouguet (2001) proposes the pyramidal Lucas-Kanade tracker. It consists on building image levels where each level contains a re-sampled image. Each image has a quarter of the immediately prior level and its smoothly quantified to avoid aliasing. This approach allows large movements since it keeps the patch size for every level. Since large movements became small in a higher level, each level satisfies the condition of small displacement for standard Lucas-Kanade. So, for each level, Lucas-Kanade is computed normally. The final displacement is a combination of each displacement found.

Lucas-Kanade and its pyramidal version output a vector field. In our case, they output the set G' of terminal points of each vector. Then, we must select the most reliable points to estimate object motion. A point $\vec{q}$ in G is reliable if its corresponding point in G' is another view of the same point, i.e., if the optical flow is correct. Usually, we test a small neighborhood to measure point reliability. The following sections present two approaches to extract these points.

3.1.2 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 patch being located in the position $\vec{q_1} = (x_1, y_1)$ of an image. NCC has low sensitivity to absolute intensity changes between reference and target images due to normalization. However, it is computationally expensive when compared to the sum of squared differences (SSD) and other common error functions.

Let T be the target patch and I be the image or the search space. NCC similarity is defined as:

$$ncc(x, y) = \frac{\sum_{x,y}(T(x, y) \cdot I(x_1 + x, y_1 + y))}{\sqrt{\sum_{x,y} T(x, y)^2 \cdot \sum_{x,y} I(x_1 + x, y_1 + y)^2}}.$$

Note that, if we compare two patches with the same resolution and fix $\vec{q} = (0, 0)$, this measure will indicate how much these patches look like each other. Therefore, it can be used to measure point reliability. Thus, let $\vec{q}$ be a point in the first image and $\vec{q_F}$ its corresponding point in the next image given by pyramidal tracker. Consider P_1 and P_2 two patches with the same resolution around $\vec{q}$ and $\vec{q_F}$, respectively. The NCC error for point $\vec{q}$ is given by $e_{NCC}(\vec{q}) = 1 - ncc(0, 0)$, that is:

$$e_{NCC}(\vec{q}) = 1 - \frac{\sum_{x,y}(P_1(x, y) \cdot P_2(x, y))}{\sqrt{\sum_{x,y} P_1(x, y)^2 \cdot \sum_{x,y} P_2(x, y)^2}}. \tag{3.1}$$

Lower NCC error means higher similarity and, consequently, higher point reliability. As proposed by Kalal et al. (2010), 50% of the points with the worst values are eliminated.

3.1.3 Forward-Backward error

Forward-backward (FB) is a dissimilarity function proposed by Kalal et al. (2010). 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 $\vec{q} = (x, y)$ be a point in the first image. By applying pyramidal tracking forward, we have the corresponding

point $\vec{q_F} = (x_F, y_F)$ in the second image. Now, by applying the tracking backward using $\vec{q_F}$ (i.e., from the second to the first image), we have the point $\vec{q_B} = (x_B, y_B)$. FB error for the point $\vec{q}$ is given by the Euclidean distance between $\vec{q}$ and $\vec{q_B}$:

$$e_{FB}(\vec{q}) = \sqrt{(x_B - x)^2 + (y_B - y)^2}. \tag{3.2}$$

FB error is depicted in Figure 3.1. 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.

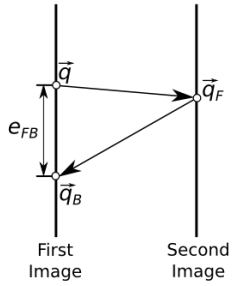

Figure 3.1: FB error for point $\vec{q}$. Adapted from Kalal et al. (2010).

3.1.4 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. (2010). The bounding box displacement is the median in each coordinate $\vec{d_m} = (d_{xm}, d_{ym})$. This way, the method is robust to impulse noise, i.e., instant and sparse 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 $\vec{q_{11}} = (x_{11}, y_{11})$ and $\vec{q_{12}} = (x_{12}, y_{12})$ be two points in the first image and $\vec{q_{21}} = (x_{21}, y_{21})$ and $\vec{q_{22}} = (x_{22}, y_{22})$ their corresponding points in the second image. The distance ratio is given by:

$$dist_{ratio}(\vec{q_{11}}, \vec{q_{12}}) = \frac{\sqrt{(x_{12} - x_{11})^2 + (y_{12} - y_{11})^2}}{\sqrt{(x_{22} - x_{21})^2 + (y_{22} - y_{21})^2}}.$$

The bounding box scale factor s_m is the median of $dist_{ratio}$. 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 B_i, the displacement vector $\vec{d_m^i} = (d_{xm}^i, d_{ym}^i)$ and the scale factor s_m^i, the coordinates of the new bounding box B_{i+1} are calculated from:

$$\begin{cases} x_{min}^{i+1} = x_{min}^i + d_{xm}^i + \frac{w^i}{2}(1 - s_m^i) \\ y_{min}^{i+1} = y_{min}^i + d_{ym}^i + \frac{h^i}{2}(1 - s_m^i) \\ x_{max}^{i+1} = x_{max}^i + d_{xm}^i - \frac{w^i}{2}(1 - s_m^i) \\ y_{max}^{i+1} = y_{max}^i + d_{ym}^i - \frac{h^i}{2}(1 - s_m^i) \end{cases},$$

where $w^i = x_{max}^i - x_{min}^i + 1$ and $h^i = y_{max}^i - y_{min}^i + 1$ are the width and height of B_i respectively. We can show that the new width is given by $w^{i+1} = w^i \cdot s_m^i$:

$$\begin{aligned} w^{i+1} &= x_{max}^{i+1} - x_{min}^{i+1} + 1 \\ w^{i+1} &= x_{max}^i + d_{xm}^i - \frac{w^i}{2}(1 - s_m^i) - x_{min}^i - d_{xm}^i - \frac{w^i}{2}(1 - s_m^i) + 1 \\ w^{i+1} &= x_{max}^i - x_{min}^i + 1 - w^i(1 - s_m^i) \\ w^{i+1} &= w^i - w^i + w^i \cdot s_m^i \\ w^{i+1} &= w^i \cdot s_m^i. \end{aligned}$$

Similarly, $h^{i+1} = h^i \cdot s_m^i$. This way, the scale is applied around the bounding box center and the center position is kept by this transformation. We define upper and lower limits of bounding box size. If the size is out of this limit or the bounding box is completely out of frame, we set an invalid value for the coordinates. If the bounding box is partially out of the scene, we estimate displacement vector and scale factor from the remaining points.

3.2 Detector

The detector presented here is an adaptation from Kalal et al. (2012) without the learning component. On TLD framework, the detector is a semi-supervised classifier retrained at runtime with the addition of some tracker responses in the training set chosen by the learning component. Here, the training stage is similar to TLD, but is performed only once. This way, we intend to evaluate detectors limitations without retraining.

3.2.1 Training

In this stage, detector learn the object appearance. It receives as input a unique bounding box containing the interest object and produce new samples from it. Then, it trains the classifier with these samples. The main algorithm is presented below.

Algorithm 2: Training.

Data: Image I_1, Bounding Box B_1.
Result: Object model M, Ensemble classifier C.
begin
  $Positive \leftarrow \text{Patch}(I_1, B_1) \cup \text{ClosestPatches}(I_1, B_1)$;
  $Positive \leftarrow Positive \cup \text{Warped}(Positive)$;
  $Negative \leftarrow \text{FarthestPatches}(I_1, B_1)$;
  $F \leftarrow \text{SetFerns}()$;
  foreach $C_i \in C$ **do**
    foreach $P \in Positive$ **do**
      Normalize(P);
      $code \leftarrow \text{GetCode}(P, F_i)$;
      $C_i[code].positive++$;
    end foreach
    foreach $N \in Negative$ **do**
      Normalize(N);
      $code \leftarrow \text{GetCode}(N, F_i)$;
      $C_i[code].negative++$;
    end foreach
    foreach *code* **do**
      $C_i[code].posterior = \frac{C_i[code].positive}{C_i[code].positive + C_i[code].negative}$;
    end foreach
  end foreach
  $M \leftarrow Positive \cup Negative$;
end

The training stage is composed of two main steps: samples selection and generation, and posteriors initialization. First, we select surrounding patches to increase the object model and generate more views from them as Kalal et al. (2012). The patch under the initial bounding box B_1 is the first element in the model. The closest patches in the image are labeled as positive examples (*ClosestPatches()*) and the farthest are labeled as negative examples (*FarthestPatches()*). Here, patches are chosen as depicted in Figure 3.2. These patches have the same resolution of the initial patch. From positive samples, we generate more examples through affine transformations (*Warped()*). We apply separately $\pm 10°$ in-plane rotation, $\pm 5\%$ scaling and $\pm 1\%$ translation in each direction (Figure 3.3).

Thus, we generate 8 warped patches for each positive sample, totaling 72 positive patches and 8 negative patches.

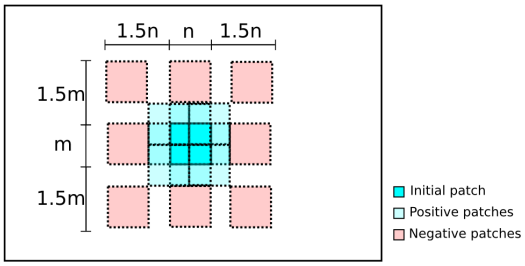

Figure 3.2: Positive and negative samples in a frame for the object model.

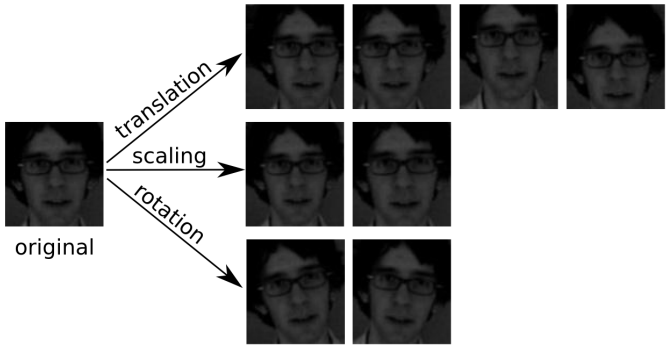

Figure 3.3: Warped images from david.

The ensemble classifier used on TLD framework is similar to the one presented in Subsection 2.2.2. However, it does not make intensity comparisons in the patch surrounding a feature point. Instead, it makes comparisons in the whole candidate patch. Moreover, it is a binary classifier: the patch may be positive or negative.

The binary tests are randomly chosen, permuted and equally divided into the ferns in *setFerns()* function. Each test is generated for a 15×15 patch. For this reason, each patch is normalized in *Normalize()* function. Each normalized patch of the model is evaluated by a fern resulting in a code (*GetCode()*). The posterior probability $P(Y = 1|f^i_{code})$ for each fern F_i is initialized as follows:

$$P(Y = 1|f^i_{code}) = \frac{p^i}{n^i + p^i}.$$

It represents the probability of a patch P with code f_{code} being positive. Y is a random variable that represents the class and can take on the values 1 for positive or -1 for negative patches. p^i is the number of positive training patches which has the code f^i_{code} with the binary tests of F_i; n^i is the number of negative patches which has the same code.

3.2.2 Detection

Similarity measures: At runtime, the detector must compare a candidate patch with the learned model. For this purpose, Kalal proposes six similarity measures. For these similarity measures, we consider normalized resolution. Thus, let P and P' be two 15×15 patches and M the object model given by $M = \{P_1^+, \cdots, P_v^+, P_1^-, \cdots, P_w^-\}$. The number of positive patches is given by v and negative is given by w.

- Patch similarity: $s(P, P') = \frac{(ncc(0,0)+1)}{2}$.
- Positive similarity: $s^+(P, M) = \max_{P_i^+ \in M} s(P, P_i^+)$.
- Negative similarity: $s^-(P, M) = \max_{P_i^- \in M} s(P, P_i^-)$.
- Earliest similarity: $s^+_{50\%}(P, M) = \max_{P_i^+ \in M \wedge i < \frac{v}{2}} s(P, P_i^+)$.
- Relative similarity: $s^r(P, M) = \frac{s^+(P,M)}{s^+(P,M)+s^-(P,M)}$.
- Conservative similarity: $s^c(P, M) = \frac{s^+_{50\%}(P,M)}{s^+_{50\%}(P,M)+s^-(P,M)}$.

Detection: The detector pseudo-code is given below:

Algorithm 3: Detector.

Data: Video $V = \{I_1, \cdots, I_k\}$, Bounding Box B_1.

Result: Trajectory $T = \{B_1, \cdots, B_k\}$.

begin

$(M, C) \leftarrow \text{Training}(I_1, B_1)$;

foreach $I_i \in V - \{I_1\}$ **do**

$P \leftarrow \text{GeneratePatches}(I_i)$;

$P \leftarrow \text{PatchVariance}(P, P_1^+)$;

$P \leftarrow \text{Normalize}(P)$;

$P \leftarrow \text{PatchEnsemble}(P, C)$;

$P \leftarrow \text{NearestNeighbor}(P, M)$;

$B_i \leftarrow \text{MostConfident}(P, M)$;

end foreach

end

The detector receives a video and an initial bounding box containing the object and trains the classifier using the bounding box. For each frame, it must decide if the object is visible. For this purpose, we generate many candidate patches that are going to be examined by the classifier (the scanning grid approach) in *GeneratePatches()* function. Each patch has the same aspect ratio than the original bounding box and is generated by scale and translation transformations of it. The minimum size is 20 pixels and maximum is the smallest of image width and height; the shift steps are $0.1n$ and $0.1m$ for each direction, and the scale step is 1.2. The classifier consists of four steps, each one eliminates several candidates.

The first step of the cascaded classifier is the variance filter. Patches with variance value smaller than 50% of the original patch variance are filtered out. The variance value for a patch P is given by $V(P) = E[P^2] - E[P]^2$, where $E[f(P)]$ is the expected value and is computed as follows:

$$E[f(P)] = \sum_{\vec{q} \in P} \frac{f(P(\vec{q}))}{\mid P \mid},$$

where $\mid P \mid$ is the number of points in P. Recall that we define patch and integral images in Chapter 2. Consider $f_1(I(\vec{q})) = I(\vec{q})$, $f_2(I(\vec{q})) = I(\vec{q})^2$ two functions over I values, and

I_{int_1}, I_{int_2} its respective integral image. For any patch, $E[P]$ and $E[P^2]$ can be calculated in a constant time. Let $q_1 = (x_1, y_1)$ and $q_2 = (x_2, y_2)$ be two points which define P. So, $E[P]$ is given by:

$$E[P] = \frac{I_{int_1}(x_2, y_2) - I_{int_1}(x_2, y_1) - I_{int_1}(x_1, y_2) + I_{int_1}(x_1, y_1)}{\mid P \mid},$$

and $E[P^2]$ is given by:

$$E[P^2] = \frac{I_{int_2}(x_2, y_2) - I_{int_2}(x_2, y_1) - I_{int_2}(x_1, y_2) + I_{int_2}(x_1, y_1)}{\mid P \mid}.$$

Here, the detector assumes that the object has a high variance value and candidates with less than 50% of this value can be filtered out. This stage removes homogeneous regions such as wall and sky.

The second stage is the classification by ensemble trained offline. A remaining candidate P is normalized and evaluated by each fern resulting in a set of probabilities $P(Y = 1 \mid f^i_{code}(P))$. If the average of probabilities is greater than 50%, P passes to the next stage. Otherwise, it is filtered out. This stage removes patches with different patterns.

The third stage is the nearest neighbor classification. If a normalized candidate patch is closer to positive patches than negative, it is classified as positive. For this purpose, we use the relative similarity $s^r(P)$. If the relative similarity of a patch is greater than 0.5 then it is classified as positive, as proposed by Kalal et al. (2012).

The last stage receives the remaining patches of the nearest neighbor classifier and output only one bounding box. In TLD framework, it selects the patch with the greatest conservative similarity $s^c(P)$. This measure is an interesting approach for TLD since the tracker feeds the training set. The tracker learns new appearances, however the first one is the most reliable appearance because it is given by the user. Here, we use the highest relative similarity since the detector is not retrained.

At the beginning of any stage, if there is no remaining patches, we consider the object as not visible. The cascaded classifier is illustrated in Figure 3.4 from Kalal et al. (2012).

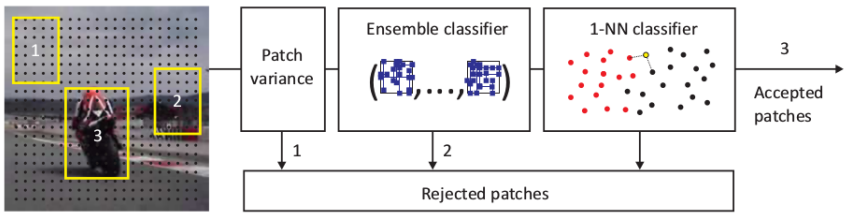

Figure 3.4: Cascaded classifier from Kalal et al. (2012).

3.2.3 Final response

For each frame of the video, the tracker outputs a bounding box and the detector outputs another. The final response of the long-term tracker is the bounding box with the highest relative similarity. We set the lowest relative similarity (zero) for a bounding box with invalid values, i.e, a not visible response. If the similarity of tracker response is less than 0.5, it receives invalid values.

4 Experimental Results

4.1 Methodology

4.1.1 TLD dataset

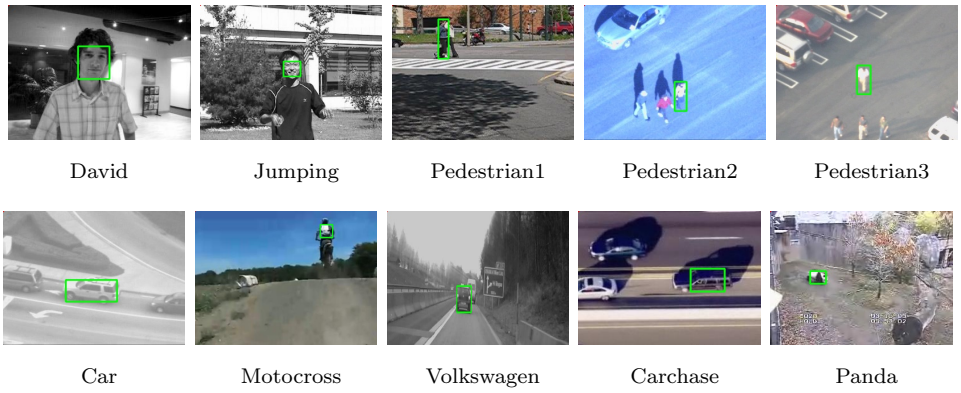

Figure 4.1: TLD sequences (Kalal et al., 2010).

To evaluate the tracker, we use TLD dataset proposed by Kalal et al. (2010). It consists of 10 sequences: david, jumping, pedestrian1, pedestrian2, pedestrian3, car, motocross, volkswagen, carchase and panda (Figure 4.1). Each sequence contains common challenges for long-term tracking (Table 4.1) as illumination changes and long-term sequences. The dataset also includes the initial bounding box and the ground truth trajectory for each sequence. If the object is not visible (out of the scene), the *nan* flag is assigned for each bounding box coordinate. More than 50% of occlusion or more than 90 degrees of out-of-plane rotation are also considered as not visible.

4.1.2 Quality metrics

Overlap measure: Overlap measure is a similarity function to compare bounding boxes. Let B_1 and B_2 be two bounding boxes, the overlap measure $o(B_1, B_2)$ is given by:

$$o(B_1, B_2) = \frac{B_1 \cap B_2}{B_1 \cup B_2},$$

Table 4.1: TLD dataset (Kalal et al., 2010).

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

where $B_1 \cap B_2$ is the intersection area and $B_1 \cup B_2$ is the union area (Figure 4.2). A bounding box is considered correct if the overlap measure between the bounding box and the ground truth is greater than 25%.

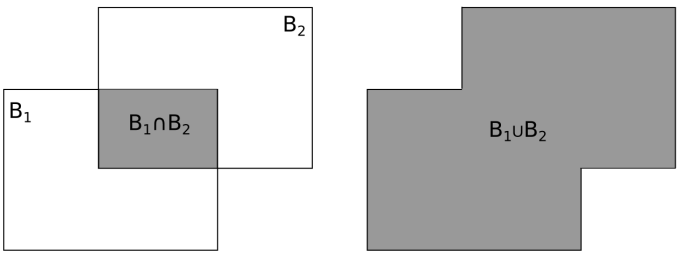

Figure 4.2: Overlap measure.

For trajectories evaluation, we use two measures. The first measure is used for pure trackers and is given by the number of correct predictions until the first mistake. This is used because the tracker might recover the object if it goes back for the same position. The second measure is used for detectors and long-term trackers, and is given by the total number of correct predictions.

4.2 Results and discussion

4.2.1 Tracker results

In this first experiment, we evaluate the tracker presented in Subsection 3.1. It is important to note that since the tracker uses the previous position, it cannot recover from a full occlusion. This way, the maximum number of correct frames for sequences from four to

ten is the index of the second time that the object appears in the scene, i.e., right after the first occlusion (Table 4.2).

Table 4.2: First full occlusion in TLD dataset.

Name	Frames	End of the first full occlusion
4. Pedestrian 2	338	39
5. Pedestrian 3	184	80
6. Car	945	566
7. Motocross	2665	33
8. Volkswagen	8576	277
9. Carchase	9928	168
10. Panda	3000	1284

As Kalal et al. (2012), we estimate optical flow using pyramidal Lucas-Kanade with 2 levels in a 10×10 grid within the bounding box. We evaluate the descriptor varying window size and filter. In addition to presented filters, we also conduct an experiment without any filter, i.e., every point of the grid is used in median flow stage. We test each filter with the following w values: $\{1, 3, 7, 10\}$. These results are shown in Tables 4.3, 4.5 and 4.4. Best results are indicated by bold font.

Table 4.3: No filter results.

Name	Frames	1	3	7	10
1. David	761	12	465	**761**	**761**
2. Jumping	313	13	15	**95**	**95**
3. Pedestrian 1	140	1	**13**	8	5
4. Pedestrian 2	338	9	**33**	**33**	**33**
5. Pedestrian 3	184	28	**52**	**52**	**52**
6. Car	945	64	246	**501**	**501**
7. Motocross	2665	1	1	**16**	**16**
8. Volkswagen	8576	3	274	**275**	**275**
9. Carchase	9928	11	68	**164**	**164**
10. Panda	3000	27	82	94	**98**

Table 4.4: FB filter results.

Name	Frames	1	3	7	10
1. David	761	13	**761**	**761**	**761**
2. Jumping	313	15	30	**36**	**36**
3. Pedestrian 1	140	2	5	**10**	**10**
4. Pedestrian 2	338	**33**	29	**33**	**33**
5. Pedestrian 3	184	**52**	**52**	**52**	**52**
6. Car	945	**510**	**510**	436	406
7. Motocross	2665	1	1	11	**16**
8. Volkswagen	8576	1	**275**	**275**	**275**
9. Carchase	9928	68	**164**	**164**	**164**
10. Panda	3000	53	53	64	**78**

Note that, for most results, the number of correct frames increases along with the window size. This occurs because the tracker produces a more homogeneous vector

Table 4.5: NCC filter results.

Name	Frames	1	3	7	10
1. David	761	13	**761**	**761**	**761**
2. Jumping	313	15	35	**218**	95
3. Pedestrian 1	140	7	**15**	12	6
4. Pedestrian 2	338	**33**	**33**	**33**	**33**
5. Pedestrian 3	184	**52**	**52**	**52**	**52**
6. Car	945	**510**	**510**	**510**	**510**
7. Motocross	2665	1	1	16	**29**
8. Volkswagen	8576	1	**275**	**275**	**275**
9. Carchase	9928	123	**164**	**164**	**164**
10. Panda	3000	79	88	94	**98**

field and median flow is able to generalize the movement. Moreover, bigger windows can handle illumination variations. For this reason, the performance increases drastically in the david sequence. As depicted in Figure 4.3, the david sequence has large variation in illumination.

First frame. Last frame.

Figure 4.3: Illumination change in david.

Observing the Tables 4.3, 4.5 and 4.4, we can also see that NCC and FB filters are able to select most reliable points improving the performance before the version without filter. NCC has the best performance of all with window size $w = 7$.

4.2.2 Detector results

Here, we show the detector results. We vary the number of ferns and features per fern. The nearest classifier parameter is kept with 0.5. The results are shown in Table 4.6.

Note that fewer ferns generally improves the performance. This occurs because more ferns is likely to have more dependencies between comparisons. The same occurs with bigger number of comparisons. The best parameters are 8 ferns and 14 comparisons per fern.

Tracker $\times$ Detector: Since detector are not retrained, it uses the first view of the object through the whole video. For this reason, the performance in david sequence

Table 4.6: Detector results.

Name	Frames	14 comparison per fern			20 comparisons per fern		
		8 ferns	12 ferns	15 ferns	8 ferns	12 ferns	15 ferns
1. David	761	**39**	37	35	38	32	29
2. Jumping	313	60	**63**	**63**	51	50	48
3. Pedestrian 1	140	23	26	**29**	14	16	16
4. Pedestrian 2	338	**113**	102	102	79	79	77
5. Pedestrian 3	184	70	72	**75**	60	63	62
6. Car	945	**593**	579	578	474	454	421
7. Motocross	2665	622	609	609	628	668	**687**
8. Volkswagen	8576	**592**	525	431	208	167	142
9. Carchase	9928	705	712	717	770	837	**863**
10. Panda	3000	232	205	181	176	265	**287**

decreases drastically. At the same time, it increases the performance in several sequences because it can recover the object. However, the detector is much slower than tracker, achieving about 4.14 fps against 356.94 fps.

Cascaded classifier: Figure 4.4 presents a qualitative result of cascaded classifier. We show each step of the classifier and the final response. The first step filters out most of street patches considering variance value. The second step returns patches which have the same pattern as the person of interest. The last step selects most confident patch correctly.

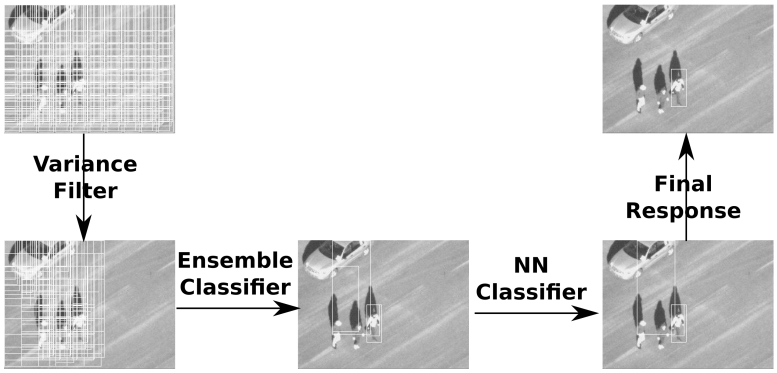

Figure 4.4: A successful classification on pedestrian2 sequence.

4.2.3 Long-term tracker results

The last experiment is the combination of tracker and detector as presented in Subsection 3.2.3. Results are shown in 4.7 together with the best results of each component.

Table 4.7: Detector results.

Name	Frames	Tracker	Detector	Long-term tracker
1. David	761	**761**	39	36
2. Jumping	313	**218**	60	55
3. Pedestrian 1	140	12	23	**28**
4. Pedestrian 2	338	33	**113**	100
5. Pedestrian 3	184	52	70	**130**
6. Car	945	510	593	**600**
7. Motocross	2665	16	**622**	455
8. Volkswagen	8576	275	592	**653**
9. Carchase	9928	164	705	**758**
10. Panda	3000	94	232	**279**

We can see that the final result is closer to the detector result. This can be explained by the similarity function used to generate the final response. Since the model is based on first object appearance and it is not updated, after some time the detector tends to always win because the tracker accept new appearances.

5 Conclusion

This work presents the study and implementation of a simplified long-term tracker from TLD (Kalal et al., 2012). For this purpose, we present basic concepts in Chapter 2 which allow us to classify trackers and evaluate their limitations. We present the median flow tracker and cascaded detector individually in Chapter 3 and the integration method. Each one has a set of parameters that is tested in Chapter 4.

It is important to highlight that this work has no intent to compare neither compete with state-of-the-art trackers. Instead, our goal is to analyze components and system limitations without retraining.

We show that tracker is faster than detector and the long-term tracker, besides being less sensitive to changes in object appearance. However, it cannot handle full occlusion by itself and tends to fail. Detector has a good performance for most of the sequences, but it is slower and has a bad performance in objects with high variation in appearance. This suggests that the combination of both improves the accuracy. However, as we show, this task requires detector retraining to learn new appearances.

Future works: Every component presented here runs sequentially. Since the detector has a high computational cost, an interesting improvement would be the paralelization of this component. Another improvement would be the generation of synthetic patches with illumination changes for the detector training set.

Bibliography

Amit, Y. **2D Object Detection and Recognition: Models, Algorithms, and Networks**. MIT Press, 2002.

Bay, H.; Tuytelaars, T. ; Gool, L. V. **Surf: Speeded up robust features**. In: Computer Vision–ECCV 2006, p. 404–417. Springer, 2006.

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

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

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: International Conference on Pattern Recognition (ICPR), p. 23–26. IEEE, 2010.

Kalal, Z.; Mikolajczyk, K. ; Matas, J. Tracking-learning-detection. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v.34, n.7, 2012.

Lepetit, V.; Pilet, J. ; Fua, P. **Point matching as a classification problem for fast and robust object pose estimation**. In: Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on, volume 2, p. II–244. IEEE, 2004.

Lepetit, V.; Fua, P. Keypoint recognition using randomized trees. **Pattern Analysis and Machine Intelligence, IEEE Transactions on**, v.28, n.9, p. 1465–1479, 2006.

Lowe, D. G. **Object recognition from local scale-invariant features**. In: Computer vision, 1999. The proceedings of the seventh IEEE international conference on, volume 2, p. 1150–1157. IEEE, 1999.

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

Oikawa, M. A.; Taketomi, T.; Yamamoto, G.; Fujisawa, M.; Amano, T.; Miyazaki, J. ; Kato, H. A model-based tracking framework for textureless 3d rigid curved objects. **SBC Journal on 3D Interactive Systems**, v.3, n.2, 2012.

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

Pernici, F.; Bimbo, A. D. Object tracking by oversampling local features. 2013.

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