HAL
open science

Object representation in local feature spaces : application to real-time tracking and detection

Antoine Tran

To cite this version:

Antoine Tran. Object representation in local feature spaces : application to real-time tracking and detection. Computer Vision and Pattern Recognition [cs.CV]. Université Paris Saclay (COmUE), 2017. English. NNT : 2017SACLY010 . tel-01712041

HAL Id: tel-01712041

https://pastel.hal.science/tel-01712041v1

Submitted on 19 Feb 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

NNT : 2017SACLS010

THÈSE DE DOCTORAT
DE L'UNIVERSITÉ PARIS-SACLAY
PRÉPARÉE À L'ENSTA PARISTECH

Ecole doctorale n°573

Ecole Doctorale Interfaces

Spécialité de doctorat: Informatique

par

M. ANTOINE TRAN

Représentation d'objets dans des espaces de caractéristiques locales : application à la poursuite de cibles temps-réel et à la détection

Thèse présentée et soutenue à Palaiseau, le 25 octobre 2017.

Composition du Jury :

Mme. CATHERINE ACHARD, Maître de Conferences, Université Pierre-et-Marie-Curie, Présidente du jury, Rapporteur

M. SERGE MIGUET, Professeur, Université Lumière Lyon 2, Rapporteur

M. STEPHANE HERBIN, Ingénieur, ONERA, Examinateur

Mme. MARIE-VÉRONIQUE SERFATY, Ingénieur, DGA, Examinatrice

M. ANTOINE MANZANERA, Enseignant-Chercheur, ENSTA ParisTech, Directeur de thèse

Acknowledgements

Il est maintenant temps de conclure la rédaction de ce manuscrit. Ayant énormément apprécié ces nombreuses années à l'U2IS, c'est avec un certain plaisir que je rédige ces quelques lignes pour remercier les personnes que j'ai cotoyé durant ces années.

La première personne à remercier est évidemment mon directeur de thèse, Antoine Manzanera. Sa disponibilité, sa patience et sa gentillesse en toutes circonstances, depuis l'entretien, fait malgré la neige gênant les transports menant à l'ENSTA, jusqu'aux nombreuses lectures, relectures de ce manuscrit de thèse en font un directeur de thèse exemplaire, avec qui j'ai pris énormément plaisir à travailler, à discuter de sujets aussi bien scientifiques que non scientifiques durant de nombreuses heures. Je tiens aussi à le remercier pour la prolongation de contrat qui a permis finir cette thèse dans de bonnes conditions. Cette thèse n'aurait pas vu le jour sans la Direction Générale de l'Armement, qui a accepté de financer ma thèse pendant 3 ans. Enfin, je remercie aussi toutes les personnes qui ont composé mon jury de thèse : les rapporteurs, Mme. Catherine Achard et M. Serge Miguet, ainsi que les examinateurs, Mme. Marie-Véronique Serfaty et M. Stéphane Herbin.

La thèse est une épreuve difficile, composée de périodes compliquées aussi bien scientifiques que morales. A ce niveau, la bonne ambiance du laboratoire, et les différentes personnes avec qui j'ai pris plaisir à discuter dans le couloir et devant un café ont fortement contribué à la réussite de cette thèse. Je tiens donc à les remercier. Je commencerai par remercier mes collègues de bureau : Daniela, qui peut maintenant occuper mon espace avec des plantes, Louise, pour les longues discussions qu'on a pu avoir sur des sujets divers et variés, les actions faites pour rendre la vie du laboratoire plus agréable et l'aide et le soutien apportés durant la rédaction, malgré sa présence partielle au labo. Je remercie aussi Matthieu, qui a bien laissé sa trace et nous a bien simplifié la vie en nous facilitant l'accès à Internet. Une petite pensée aussi pour Fabio et Phuong, anciens post-docs qui auront trouvé des postes dans des lieux plus ensoleillés que Palaiseau.

Une grande partie des bons moments passés au labo sont liés aux différents doctorants rencontrés durant ma thèse. Je pense notamment aux réunions de doctorants co-organisés et co-animés avec Pauline, aux jeux de plateau découverts avec Gennaro, aux thés du matin et pauses

de 16h avec Céline, aux intrusions et vols de post-it du bureau d'Adrien, aux montages électroniques de Pierre-Henri qui ne cesseront jamais de m'impressionner, à la présence de Clément M. y compris après son départ, ainsi qu'à l'humour particulier de Clément P. (que je reverrai avec plaisir dans la suite de mon parcours professionnel).

Je remercie aussi les post-docs qui ont fait part de leur sagesse, tout au long de cette thèse : Adina, pour toute la patience accordée depuis son arrivée, en m'écoutant m'énerver sur la rédaction et avec qui j'ai eu de grandes et (très) longues conversations, François F., avec qui j'ai présenté de nombreuses fois les travaux du laboratoire aux extérieurs, Julien, qui a souvent pris le temps de partager sa vision de la thèse et de la recherche, Olivier, le nouveau membre de l'équipe course à pied de l'U2IS, Taha et sa bonne humeur, et Natalia, avec qui j'ai malheureusement pu constater mes faiblesses en espagnol.

Enfin, je remercie aussi les permanents et ingénieurs de l'U2IS, en particulier Alexandre (avec qui j'espère pouvoir faire de nouvelles courses dans le futur), Thibault (pour avoir réparé les charnières de mon PC), François P. (pour avoir réparé l'alimentation de ce même PC) et Catherine, pour m'avoir sauvé de toutes les démarches administratives.

De manière générale, je souhaite remercier l'ensemble des personnes de l'U2IS, qui m'ont vu (et revu) dans les couloirs du laboratoire ou près de la machine à café, qui ont rendu mon séjour très agréable.

Contents

Acknowledgements **3**

1 Introduction **1**
1.1 Scope of the thesis 3
1.2 Contributions 5
1.3 Outline of the thesis 6

2 Object representation **9**
2.1 Visual Features 10
2.1.1 Color-Based features 11
2.1.1.1 Color space 12
2.1.1.2 Color Histogram 15
2.1.1.3 Higher Level of representation 18
2.1.1.4 Conclusion 21
2.1.2 Shape-Based Representation 22
2.1.2.1 Mathematical context 22
2.1.2.2 Sobel filter 22
2.1.2.3 Local Jet space 24
2.1.2.4 Higher level features 30
2.1.2.5 Conclusion 33
2.1.3 Conclusion of the section 33
2.2 Hough Transform 34
2.2.1 History of the Hough Transform 35
2.2.2 General formulation 36
2.2.2.1 Variant of the Hough Transform 39
2.2.3 Generalized Hough Transform 40
2.2.4 Applications of Hough Transforms in computer vision 43
2.3 Conclusion 44

3 Object Tracking **47**
3.1 Definition 50
3.1.1 Tracking conditions 53
3.1.2 Difficulties 54
3.1.3 Conclusion 55
3.2 Literature review 56
3.2.1 State-of-the-art 58
3.2.2 Hough Transform for Object Tracking 64
3.3 Combining color histogram and Gradient for tracking . . . 67

3.3.1 Backprojection map 68
3.3.2 Combining GHT and Particle Filter 69
3.3.3 Transitional tracker 73
3.3.4 Final tracker 77
3.3.4.1 Position estimation 79
3.3.4.2 Scale and orientation estimations 81
3.3.4.3 Updating model 82
3.3.4.4 Conclusion 83
3.4 Results 84
3.4.1 Implementation details 84
3.4.1.1 Optimization 84
3.4.2 VOT datasets 86
3.4.2.1 History of the VOT Challenge 86
3.4.2.2 Parameter details 88
3.4.2.3 VOT2014 92
3.4.2.4 VOT15 95
3.4.2.5 Analysis of results 98
3.5 Conclusion 111

4 Object Detection 115
4.1 Literature review 117
4.1.1 Object classification, detection, recognition 117
4.1.2 Hough detectors 119
4.2 Hough Forest 120
4.2.1 Forest training 120
4.2.1.1 Generating training set 121
4.2.1.2 Training tree 122
4.2.1.3 Implementation details 125
4.2.1.4 Detection 125
4.2.2 Extensions of the Hough Forest 126
4.3 Our contributions 128
4.3.1 Patch generation 128
4.3.2 Node training 129
4.4 Experiments 132
4.4.1 Evaluation method 132
4.4.2 UIUC Cars 134
4.4.3 TUD Pedestrian 135
4.5 Conclusion 140

5 Perspectives and Conclusion 143
5.1 Conclusion 143
5.2 Perspectives 144
5.2.1 Raspberry Pi 3 Implementation 145
5.2.1.1 Global perspectives 146

A Per sequence results on VOT2015 for different color spaces 149

B Per sequence results on VOT2015 for different derivatives scales 155

Bibliography 159

List of Figures

1.1 Traditional chain of computer vision 1
1.2 Tracking process from *ironman* sequence 4
1.3 Pedestrian detection (TUD-Pedestrians dataset) 5

2.1 First frame from *surfing* sequence [KPL+] 11
2.2 Image of the first line splitted into RGB, HSV and Lab color spaces (respectively second, third and fourth lines) . 16
2.3 Histogram taken from the first frame of *bolt* sequence [KPL+] 17
2.4 Two different flags with the same color distribution 17
2.5 Superpixel segmentation obtained from *ball* sequence [KPL+] with 400 superpixels. 20
2.6 Color attribute of frame 0 of *marching* sequence 21
2.7 Sobel in x and y applied in frame from *torus* sequence [KPL+] 23
2.8 Magnitude and orientation from *torus* image. For a better visualization, we displayed only orientation for pixels with high gradient magnitude 23
2.9 Gradient magnitude at different scales 26
2.10 Junctions and blobs at different scales 27
2.11 Griffin norm computed at scales $\{1, 2, 4\}$ 28
2.12 Corners detected in *lego1* sequence 31
2.13 Fast keypoint schematic. The central point is detected as a keypoint by the FAST algorithm: in the circle of circumference 16, there is one large contiguous set of pixels brighter than the central point. 31
2.14 Summary of different presented features 34
2.15 Figure from Hough's patent. Each point from lines of the upper part generates one line in the lower part. Then, all aligned points from the upper part generates a beam of concurrent lines in the lower one 36
2.16 Parametrization of the blue line using Duda [DH72] parameter set (r, θ) 37
2.17 Line Hough Transform on three points: $A(1, 1)$, $B = (1.5, 0.5)$ and $C = (2.5, -0.5)$ 38
2.18 Lines detection using Hough Transform 38
2.19 Circles detection 38

2.20 Building the R-Table. On the left part, a contour image, with a reference point r. For each pixel p from the contour, the gradient orientation θ_p is computed, and contributes to the R-Table in the entry $R(\theta_p)$, by the displacement $\overrightarrow{pr}$ . . 40
2.21 Generalized Hough Transform on the contour image (on the left), and the Hough Transform on the right 41
2.22 GHT on a scene, with the squared sheep used as a model . 42

3.1 Different types of object representation 51
3.2 Diagram of a generic tracker. 52
3.3 Diagram of model-free tracker. 53
3.4 Some difficult frames from VOT2014 and VOT2015 datasets 56
3.5 Backprojection from *bag* sequence. 69
3.6 Backprojection obtained by Eq.3.11. 71
3.7 Diagram of tracker [TM15] 72
3.8 Impact of prediction map in the GHT 75
3.9 Surrounding area from *sunshade*. 75
3.10 Mapping of the blue and red areas using H_t^R (Eq 3.16) . . 76
3.11 Some non consecutive frames from *motocross* sequence . . 77
3.12 Impact of Possegger's formulation [PMB15] on the color-based confidence map 79
3.13 Diagram of position estimation (better in color) 81
3.14 Illustration of overlappping 87
3.15 Expected overlap for different values for μ_c and μ_g 90
3.16 Accuracy rank for different values for μ_c and μ_g 91
3.17 Robustness rank for different values for μ_c and μ_g 92
3.18 Expected overlap for different values of n_c and n_g 94
3.19 Accuracy rank for different values of n_c and n_g 95
3.20 Robustness rank for different values of n_c and n_g 96
3.21 Speed for different values of n_c and n_g 97
3.22 Frames from *hand2* sequence 97
3.23 Frames from *torus* sequence 100
3.24 AR plot for trackers displayed on Tab. 3.5 102
3.25 AR plot for trackers displayed on Tab. 3.6 for VOT15 . . . 104
3.26 Frames from *matrix* sequence 104
3.27 Tracking from *butterfly* sequence 105
3.28 Tracking from *iceskater2* sequence 105
3.29 Accuracy and failure for different versions of CHT 107
3.30 Frames from *ball2*, *car1* and *helicopter* and their projection into channel a and b. 109
3.31 Frames 9 and 10 from *bag*, with their color attributes mapping. 110
3.32 Color attribute from *fernando*. 110
3.33 Gradient magnitude for $\sigma \in \{2, 4, 8\}$ 111

4.1 Instances from pedestrian class 116
4.2 Patch sampling from images from UIUC dataset [AAR04]. 122

4.3 A binary tree of depth 3. 122
4.4 Patch set split into two disjoint parts 123
4.5 Failures and right detections 126
4.6 Junctions shown by circles for one landscape and two pedestrian images 129
4.7 Generating Π^a 131
4.8 Generating superpatches 131
4.9 Examples of TP, FP and FN 133
4.10 Negative and positive images from UIUC-Cars 135
4.11 ROC curve on UIUC Cars multi-scale dataset 136
4.12 One training image from TUD-pedestrian with its associated segmentation map 136
4.13 GallHOG *vs* Sobel *vs* MSDerivatives 138
4.14 Worst and best curves in different cases 139
4.15 Average ROC curves, and covariance ellipses for different threshold points 139
4.16 MSDerivatives *vs* DerivativeS1 *vs* DerivativeS2 *vs* Sobel . 140

5.1 Raspberry Pi 3 Model B 145

List of Tables

1.1 Desired properties and limitations of our approach 4

2.1 Crosier's classification, at a given scale σ 28

3.1 Robustness ranking for different couples (μ_g, μ_c) 93
3.2 Parameters set 98
3.3 Baseline results for VOT2014 (ranks over 43 candidates). CHT, CHTs and CHTf are our trackers. 99
3.4 Region noise results for VOT2014 100
3.5 Overall results for VOT2014 101
3.6 Overall results for VOT2015 (ranks go to 66) 103
3.7 Overlap and number of failure for different trackers in the sequence *sheep* 106
3.8 Number of failures per rank on Camera Motion an Illumination change 106
3.9 Expected overlap for different forms of [TM17] 107
3.10 Results for color variant trackers 108
3.11 Expected overlap for geometrical variant trackers 108
3.12 Accuracy for geometrical variant trackers 108
3.13 Overlap of road for different values of σ 111

4.1 Parameters set proposed by Gall, and ours 134

A.1 Performances by varying color space (part 1) 151
A.2 Performances by varying color space (part 2) 153

B.1 Performances by varying derivative scale 157

Chapter 1

Introduction

Computer vision aims to understand, to interpret an image or a video like (or even better than) a human being. Understanding or interpreting an image, a video, can have many forms. It can consist in detecting a specific target in an image (object detection), following the trajectory of one object (tracking), classifying one object instance with respect to different categories (recognition) or, given a set of images, selecting those fulfilling one specific condition (Content Based Image Retrieval). Those applications are now present in everyday life, such as in engine search or social network systems, surveillance systems, mobile phones, etc.

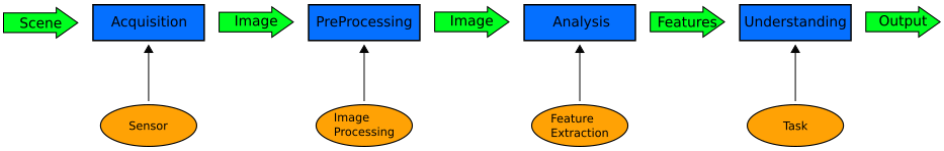


FIGURE 1.1: Traditional chain of computer vision

Traditionally, a computer vision algorithm is subdivided into different functions (see Fig. 1.1):

- *Acquisition*: given a scene to observe, a sensor (such as a camera) is capturing the scene. This step is all the more important that a bad acquisition (due for example to a low quality sensor) will spread errors in the whole computer vision chain. Depending on the scene to acquire, several constraints can appear. In medical or radar imaging, captured image can be very noisy, or image resolution may not be sufficient for the task. Acquiring the motion of fast objects (sport videos typically) implies the use of a camera able to capture decent images with very low exposure time. In all cases, the sensor is the critical element of this step

- *Pre-processing*: after the acquisition, the quality of the capture may not be sufficient for the task. In that case, a processing step is required to enhance the quality. This enhancement can take several forms: denoising, deblurring...

- *Analysis*: after processing, the captured image (or the video) is not ready yet for higher level interpretation: too much information is included in processed data. Consequently, the analysis step consists in reducing the information by extracting the features adapted to the wanted task

- *Understanding*: from the data projected into a specific feature space, the system performs the task it has been designed for. It requires an algorithm usually designed for this task. This is usually the last step of computer vision

Nowadays, this chain may no longer be relevant since, for many systems, such as smart cameras or Vision Processing Unit (VPU), the different parts may be too intimately combined to be distinguishable. In our work, we do not step out the traditional chain, but aim to reduce the computational gap between the two last steps by using unified representations and primitives that are used all along the computer vision chain. More precisely, we study two fundamental tasks in computer vision: object tracking and detection. We want to propose a unique feature space and rely on one main algorithm for these two tasks.

On the one hand, feature extraction is a popular task in computer vision, and we find in the literature many different features: keypoints, statistical features, wavelet, and more recently deep learning autoencoders. In our case, we take the decision to use only low-level and local color and shape features. More precisely, we aim work only with pixel colors and their derivatives only.

On the other hand, the understanding step consists in using a specific algorithm exploiting information represented in the designed feature space to get a high-level information, a better understanding of the scene. This high-level information can be the position of one specific target, recognizing some specific events, classifying objects.

We plan to study two different tasks: object detection and object tracking. Taken alone, both tasks lead to many applications in surveillance (dangerous object detection, following the trajectory of a threat), in robotics

(visual control) and other fields. Combined, they provide a higher-level of understanding. For example, let us consider a camera filming a bifurcation. By combining a tracker and a detector, we can determine how many pedestrians start from one entry point, and leave another one. This example suggests the main interest of our work. Indeed, autonomous systems have to perform different tasks, while being constrained by the hardware. By unifying the feature space and the algorithm, we reduce the memory consumption. Our work also presents an interest in terms of computation time. As low-level features are simpler than higher-level ones, they can be more easily implemented on any kind of system, from low-cost or embedded ones to computer clusters, and mobile phones or desktop machines.

1.1 Scope of the thesis

As mentioned previously, we aim to work with a unique feature space and algorithm, and study benefits and limitations of these hypotheses on two tasks: object tracking and object detection. In the framework of our thesis, we will suppose that sequences and images studied will be acquired from standard cameras (from phones or webcam). In this case, we discard some problematic cases that can imply image preprocessing (such as denoising).

In terms of feature space, we will work on "natural" visual features, including color and local geometry, as opposed to massive decomposition features, like wavelet, filter banks or deep learning autoencoders features. More precisely, all proposed algorithms will rely exclusively on pixel colors, spatial derivatives, and spatial aggregation mechanisms based on histograms and Hough transform. Usually, low-level features are used to build more complex ones, with higher power of representation: color attributes, keypoints, statistical features. However, those higher level features are usually slower to compute, their high power of representation might not be necessary for some tasks, and are not as flexible as low level ones (for example, keypoints cannot be found everywhere, they are particular points in images). Inversely, pixels colors and derivatives can be computed in the whole image and are adapted to parallel computation, potentially leading to a very low computation time. As the main problem is their relatively weak power of representation, one tackled problem will concern the possibility to propose effective tracker and detector based only on pixel colors and derivatives.

FIGURE 1.2: Tracking process from *ironman* sequence

In terms of representation, one algorithm has a central role in our work: the Generalized Hough Transform. Its original principle was to build a codebook to represent one contour. This codebook, indexed by gradient orientation, stores all relative positions of the contour points with respect to the center. It serves to localize the contour in a tested image. The Generalized Hough Transform has proven versatile, since many variations of the algorithm have been proposed, with regard to the feature used to index the table, to the structure of the codebook and to the parameter space. In terms of application, it has been adapted to many tasks, in particular object tracking and detection. However, the Generalized Hough Transform is not the most effective algorithm used for these tasks: Discriminative Correlation Filter and Convolutional Neural Network are state-of-the-art algorithms for tracking and detection respectively. Tab. 1.1 summarizes all benefits all weaknesses of our two hypotheses.

	Feature	Algorithm
Advantages	• Fast to compute • Good properties for parallelization	• Low memory consumption • Versatile algorithm
Limitations	Low power of representation	Not the most effective algorithm

TABLE 1.1: Desired properties and limitations of our approach

Object tracking and detection present an interest for our thesis, due to their distinct aims. In tracking, feature space is designed to estimate target trajectory as accurate and robustly as possible. This target can have any aspect (Fig. 1.2 shows one sequence found in one academic dataset [ARS08]), and in some applications (visual control notably), time consumption is critical. In detection, as feature space aims to describe a

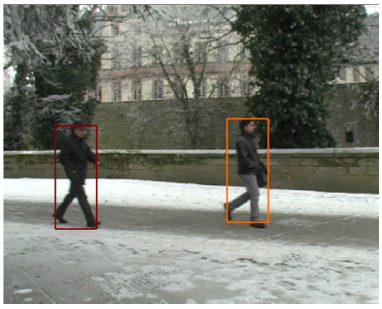


FIGURE 1.3: Pedestrian detection (TUD-Pedestrians dataset)

class of object and cope with intra-class variation, using a machine learning classifier is a possible solution (illustration of pedestrian detection Fig. 1.3). The object representation, trained with many samples, is usually heavier compared to tracking, but time consumption is also less critical. The two tasks then have different constraints in terms of representation and of computation time. Studying benefits and limits of low-level features in both cases will give us an overview of their benefits.

1.2 Contributions

Our work led to three publications in international conferences, one related to object detection, two related to object tracking:

- [TM16] Antoine Tran and Antoine Manzanera. "Fast growing Hough forest as a stable model for object detection". In: *Image Processing Theory Tools and Applications (IPTA), 2016 6th International Conference on.* IEEE. 2016, pp. 1-6

- [TM15] Antoine Tran and Antoine Manzanera. "A versatile object tracking algorithm combining Particle Filter and Generalised Hough Transform". In: *Image Processing Theory, Tools and Applications (IPTA), 2015 International Conference on.* IEEE. 2015, pp. 105-110

- [TM17] Antoine Tran and Antoine Manzanera. "Mixing Hough and Color Histogram Models for Accurate Real-Time Object Tracking" In: *International Conference on Computer Analysis of Images and Patterns* (to appear). 2017 pp. 105-110

[TM15] presents a tracker combining the Generalized Hough Transform relying on gradient features, and a Particle Filter based on color histogram. In [TM17], while still keeping the Generalized Hough Transform under the same form, we exploit color histogram to build a pixel confidence map. [TM17] has proven competitive, we earned the right to be co-author of the report dedicated to the Visual Object Tracking challenge 2017 [1]. Results, ranking, and implementation will be available in November, 2017. [TM16] deals with the Hough Forest algorithm [GL13], and proposes two contributions: one related to the training set generation, the second to the node training.

1.3 Outline of the thesis

This thesis will be divided into four parts.

Chapter. 2 will deal with object representations. Its first section will be dedicated to color features, starting from definition of color space, ending with color-based features. In the second section, we will deal with shape-based features. The first task will be to define the notion of derivative on images. Then, from these derivatives, we will see how several works have been proposed to build visual features directly computed from gradient and Laplacian, and to build higher-level ones, such as keypoint or HOG features. In these two parts, we will also present some classical tasks computed with the different presented features. The Hough Transform will then be formally defined in the third part. After an history of the algorithm, that leads to the Generalized Hough Transform, and some improvement of the original algorithm, we will illustrate the versatility of the Hough Transform by showing how it was adapted for different applications.

In Chapter. 3, we will focus on object tracking. Since in computer vision, tracking (in general) is a very popular task, we will start by defining the framework in which we are working on, and the difficulties related to object tracking. Then, we will make a literature review of methods for object tracking, by focusing on trackers related to ours and those that are comparable, in terms of performances. We will also present recent works on Hough-based trackers. In the third section, we will focus on the two proposed trackers. After detailing [TM15], we will discuss some missing

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

elements of this tracker to propose a transitional one. This transitional tracker was simplified, and finally led to [TM17], which combined accuracy, robustness and speed (one of the fastest in the literature). We will end this chapter by presenting some experimental results. By first explaining how we managed to reach a low execution time, we will continue by parameters setting. Then, results on academic datasets will be shown, together with an analysis of obtained results and the impact of low-level features.

Object detection will have the main role in Chapter. 4. As for the previous chapter, we will start by presenting the task of object detection. This presentation will be followed by a literature review in object detection, by in particular presenting state-of-the-art detectors and Hough-based ones. Then, we will present our work, by starting from an explanation of the Hough Forest algorithm [GL13], serving as a base for our detector, followed by a review of different extensions of it. We will then focus on [TM16], by detailing reasons that led us to propose such contributions. Finally, details about experimental results will be provided.

Finally, Chapter. 5 will serve as a conclusion. It will be dedicated to discussions and possible perspectives related to our work. We will notably mention the development of an autonomous system combining tracker and detector, by tackling all issues that should be solved, considering the state of our work.

Chapter 2

Object representation

Contents

2.1 Visual Features **10**
2.1.1 Color-Based features 11
2.1.2 Shape-Based Representation 22
2.1.3 Conclusion of the section 33
2.2 Hough Transform **34**
2.2.1 History of the Hough Transform 35
2.2.2 General formulation 36
2.2.3 Generalized Hough Transform 40
2.2.4 Applications of Hough Transforms in computer vision 43
2.3 Conclusion **44**

This chapter presents the framework of our work. As mentioned in the introduction, we set two limits to our work: one concerning the complexity of the feature spaces, one concerning the method to exploit those features for different tasks. Accordingly, this chapter will be divided into two parts, reflecting the two hypotheses we set:

- First, we will present some visual features used in computer vision. We will mainly focus on color-based and shape-based features (we will denote them as visual features). The aim will not to be exhaustive, but rather to see how, from very simple features (pixel color, local geometry), higher visual features were designed for computer vision tasks. All along this section, we will also present some classical computer vision tasks based on the presented features

- Second, we will deal will the *Hough Transform*. This algorithm, very popular in computer vision, has a central role in our thesis.

After presenting the history of the algorithm, starting from a patent presenting a device designed to track particles in bubble machines, then a formalization in order to detect analytical shapes and some enhancements of the original Hough Transform, we will present the extension proposed by Ballard [Bal81], the *Generalized Hough Transform*, used to detect any kind of shape. Finally, to show how versatile the algorithm is, we will make a short review of literature showing that the Hough Transform has been adapted for many applications

2.1 Visual Features

In this section, we will focus on visual features. The goal of our thesis is to study potentials and limitations of low-level representations in computer vision, for two fundamental tasks: object tracking and object detection. The main benefit of low-level features is in terms of computation time: fast to compute and easily parallelizable, they can be used for reactive algorithms. However, due to the limit in their power of representation, higher-level and more costly features have been designed, generally built from low-level ones. We will then make a review of different visual features found in the literature. Before that, we start by defining what, in our point of view, a good feature is, in machine learning, then for two tasks in computer vision.

In Machine Learning, extracting a feature from an object means creating a numerical vector supposed to represent it, to describe it. This process is called model generation. Ideally, this feature vector should fulfill several properties:

- Discriminative, to identify the observed object, and distinguish it from other ones. Mathematically, given a feature space (the space in which the vector is represented) and a good metric, the distance between two vectors of two different objects should be high

- Characteristic and intrinsic to this object, to ensure that different instances from the same object have approximately the same feature vector. Moreover, for each observation (and for each environment or context), the extracted feature vector should always be the same

- Repeatability, meaning that in noisy or degraded conditions, the obtained feature vector of the same object should not change much

One hypothesis that can be added, but is not essential, concerns the quantity of information stored in the feature vector. Ideally, this feature vector should contain the minimal amount of information necessary to describe an object. Evidently, extracted features should suit the wanted task. In object detection, features should represent the class of object as precisely as possible, while neglecting particularities of instances (for example, car's color in car detection). In object tracking, it is possible to represent the target using features able to discriminate the target from the background (for instance, tracking the surfer on Fig. 2.1 can be done using color, as the background is mostly blue, while the surfer is dark). The first part of

FIGURE 2.1: First frame from *surfing* sequence [KPL+]

this section concerns visual features: we will then make a literature review of visual features in this chapter, starting by color-based ones, and ending with shape-based ones. In both cases, we will also mention some works exploiting the different presented features to illustrates utility of these features. We will also set the limit of our thesis in terms of level of complexity of used features.

2.1.1 Color-Based features

Color-based features are among the most popular visual features used in computer vision. In the visual attention system proposed by Itti and Koch [IKN+98], color is one early feature, among intensity and orientations, used to generate saliency maps. Moreover, color is a visual feature presenting a certain robustness to some traditional geometric transformations:

in-plane rotations, scaling (robustness in terms of proportion of pixels), and translations.

However, while color is usable to describe a *specific* instance of object ("This is a *blue* car"), it is not a *generic* visual feature used to describe a class of object (All cars are not blue). Moreover, the perceived color of an object can not be clearly defined. Indeed, depending on the source of light, the perceived color can be different (lighting a green wall with a red light will not make it appear green), and too much or not enough light will drastically change the color. The color is also dependent on the sensitivity of the image acquisition system. It means that, considering hypotheses defining a good feature, color-based ones are not characteristic to the object. Consequently, they are not suitable for tasks involving high-level of semantic, and notably those requiring description of class of objects (object classification or detection).

However, color-based features still have a very important place in computer vision, as many researchers designed different color features, and are involved in many tasks: in object tracking, the main interest of color features is their ability to distinguish targets from the background (except in color camouflage case). Moreover, several authors designed some color-based high-level features, as a first step of object recognition. Then, in this section, we will present some color-based features, from very low-level (color space and histograms), to higher ones (superpixels or color attributes).

2.1.1.1 Color space

In computer vision, the first level of color-based features is extracted from the smallest element of an image: the pixel. Each pixel has a color. This color is firstly encoded into a multi (usually 3) dimensional space, called color space.

Different kinds of color space exist, but the most famous one is the RGB color space. It comes from the Human Visual System, in which colors are detected by three types of cones, each one sensitive to one specific color: Red, Green and Blue. By mixing all these three colors, in different proportions, it is possible to describe a large range of colors. Indeed, in image processing, a pixel represented in the RGB model is usually encoded by a triplet of 8-bit integers, meaning that there are $256 \times 256 \times 256 = 2^{24}$ possible combinations. However, for computer

vision tasks, one limit of this color space is the poverty of metrics to associate a visual perception of an object to the mathematical model (How red is it?). For example, a totally pure red object with be encoded by $(255, 0, 0)$, while a pure white object will be encoded $(255, 255, 255)$. Both have the same red intensity, but their colors are completely different.

Finlayson [FHX05] proposed a mathematical formula modeling illumination change and shift. Given an acquired scene for which, at a pixel p, it is initially observed a color (r_i, g_i, b_i), in case of illumination variation, this same pixel will have values (r_o, g_o, b_o) such that:

$$\begin{pmatrix} r_o \\ g_o \\ b_o \end{pmatrix} = \begin{pmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & c \end{pmatrix} \cdot \begin{pmatrix} r_i \\ g_i \\ b_i \end{pmatrix} + \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix} \tag{2.1}$$

From this equation, Van de Sande [VDSGS10] proposed to model common illumination changes, such as light intensity change (due to shadows for example), for which $a = b = c$ and $c_1 = c_2 = c_3 = 0$, or light intensity shift, indicating that a scene is illuminated with another white light source, and for which $a = b = c = 0$ and $c_1 = c_2 = c_3$.

Another popular color space is the HSV (and other similar color spaces such as HSL or HSI) color space, proposed by Smith [Smi78]. H stands for *Hue*, S for *Saturation* and V for *Value*. Given a color c encoded in the RGB space by (r, g, b), let us define $M = max(r, g, b)$, $m = min(r, g, b)$ and $C = M - m$. Then, to convert a RGB pixel to a HSV one, we use the transformation:

$$H = \begin{cases} \text{Undefined} & \text{if}(M - m) = 0 \\ 60 \cdot \frac{g-b}{C} & \text{if } M = r \\ 60 \cdot \frac{b-r}{C} + 120 & \text{if } M = g \\ 60 \cdot \frac{r-g}{C} + 240 & \text{if } M = b \end{cases} \tag{2.2}$$

$$S = \begin{cases} 0 & \text{if } M = 0 \\ \frac{M-m}{M} & \text{otherwise} \end{cases} \tag{2.3}$$

$$V = M \tag{2.4}$$

In this representation, $H \in [0, 360]$ represents the perceived color: values

close to 0 and to 360 corresponds to red, 120 to green, 240 to blue. Given Eq. 2.1 and Eq. 2.2, we can notice that the Hue is invariant to light intensity scale and shift. $S \in [0, 1]$ corresponds to the intensity of the color. Visually, it means that the smaller the saturation is, the more the color will be perceived as gray. We can also note that for $S = 0$, H is undefined, which corresponds to $r = g = b$, i.e. a grey level. It means that, in this color space, the grayscale is not properly defined. V is the value, and is also called brightness. Moreover, the smaller the saturation is, the more unreliable the Hue value is.

Finally, the last color space we will mention is the Lab color space. It belongs to the class of color-opponent space, and put in opposition the red color with the green (channel a^*), and the yellow with the blue (channel b^*). The third channel, L^*, is the luminance. According to Tkalcic in [TT+03], in the 19th century, Ewald Hering proposed the opponent colors theory, by noticing that the perceptions of certain colors are interdependent: the couple red-green, and yellow-blue. Later, still according to [TT+03], researchers supposed that from a signal in RGB acquired by the cones of eyes, the Human Visual System converts it into a signal composed of three components, each one defining a color opposition: White-Black, Red-Green and Yellow-Blue. These oppositions are the base for the Lab color space. To move from the RGB space to the Lab, the first step consists in using an intermediate one, XYZ, defined by the Commission Internationale de l'Éclairage (CIE). This is done by a linear combination:

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \mathbf{M} \cdot \begin{pmatrix} R \\ G \\ B \end{pmatrix} \tag{2.5}$$

with $\mathbf{M}$ a 3×3 matrix. Each channel of XYZ color space is supposed to represent a stimulus of the three types of cones of the Human Visual System. Then, Lab is built from XYZ in order to set L as a luminance channel, a is a channel putting in opposition the green (negative values) with the red (positive values) and the b channel to oppose blue with yellow.

This part served to present different color spaces. The first one, the RGB color space, is directly inspired by the human visual system, composed by 3 types of cones (sensitive to red, blue and green respectively) and a set of rods (sensitive to luminance). At a first glance, it does not seem suitable for computer vision task: not robust to illumination change,

and there is no direct link between the color space and the visual perception. However, it is still much used in computer vision, and especially in object tracking (we refer to Section. 3.2 for details). We also described the HSV color space. Closer to the visual perception, it quantifies three notions linked to visual perception: the observed color, the intensity of the color and the brightness. However, in this color space, grayscales are not properly defined. Finally, the Lab one, modeling the signal sent by the Human Visual System, is based on the opponent color theory: some colors such as the green and the red, are not perceived at the same time. All described color spaces are illustrated in Fig. 2.2. We will refer to Van de Sande [VDSGS10] for a richer description of the different color spaces. This presentation will be important for the rest of this document, especially for Section 3.4.2.5, in which we will evaluate impacts of different color spaces on tracking context. In the rest of this section, we will explain how to create features usable for computer vision, and present some tasks implying color features.

2.1.1.2 Color Histogram

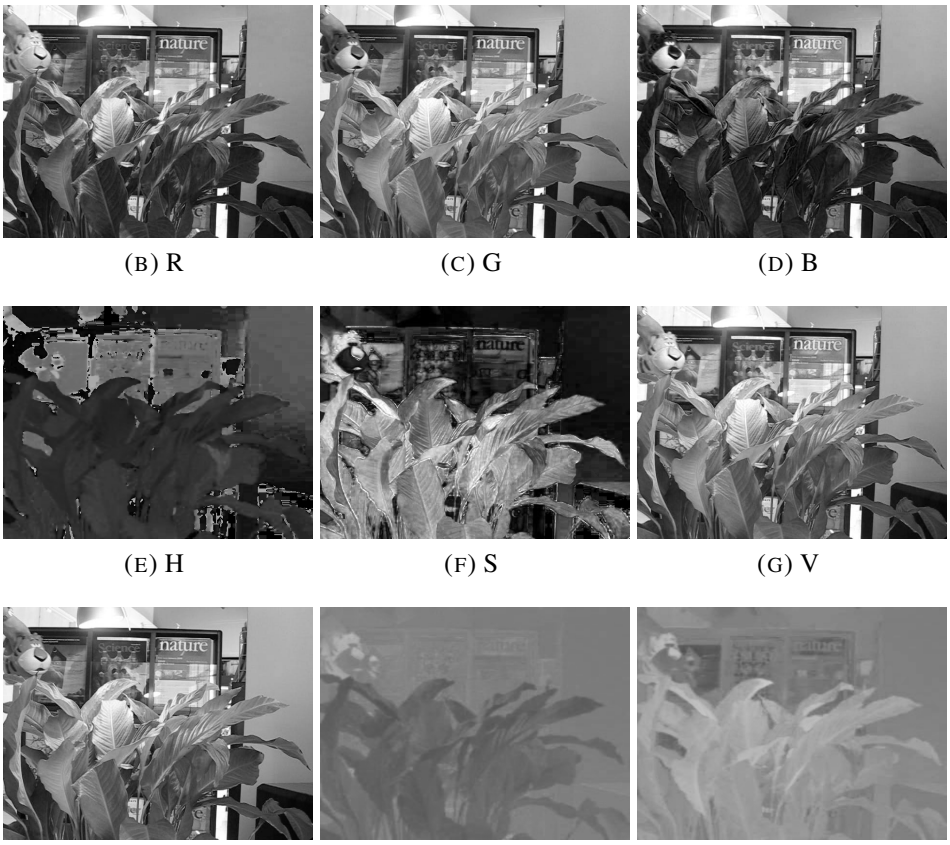
Color histogram is a very low level color-based feature. Given n_b the number of bins and an image $\mathbf{I}$, building the histogram of $\mathbf{I}$ (denoted $H_{\mathbf{I}}$) consists in, for each bin b, counting how many pixels from $\mathbf{I}$ is falling inside the bin b. Formally, for each pixel p, by denoting $\hat{\mathbf{I}}(p)$ the quantified value of $\mathbf{I}(p)$, we have, for each bin b:

$$H_{\mathbf{I}}(b) = \sum_{p} \delta(\hat{\mathbf{I}}(p) - b) \tag{2.6}$$

Fig. 2.3 shows a color histogram computed from a grayscale image. For multi-channel images, the operation is generalized by considering multi-dimensional bins.

The main advantage of using color histogram as a model for an object is the very low memory consumption (one integer per bin), and the low computation time: different methods, such as Look-Up Table (see Section. 3) or integral images [Por05], are possible to quickly evaluate an image histogram. Another advantage is made by working with normalized histogram (the sum of the values of all bins is equal to 1.0), and considering the resulting histogram as an empirical discrete probability distribution. In this condition, several metrics, based on statistics notably, exist in

(A) Input



(B) R (C) G (D) B

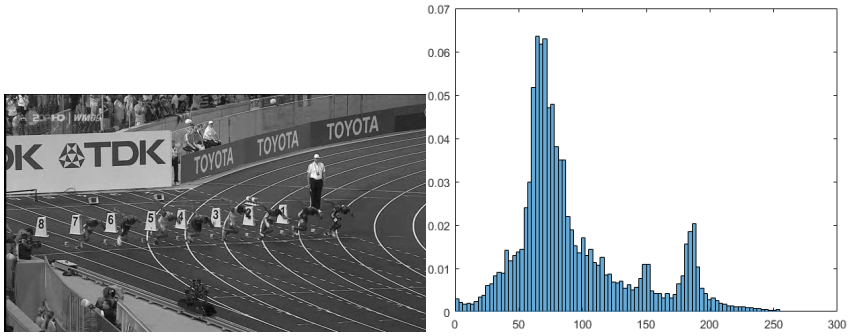
(E) H (F) S (G) V

(H) L (I) a (J) b

FIGURE 2.2: Image of the first line splitted into RGB, HSV and Lab color spaces (respectively second, third and fourth lines)

order to compare two histograms. The Bhattacharyya coefficient [Bha43] is one of the most popular metrics, and Swain [SB91] proposed a metric called *Histogram Intersection* in order to classify objects, with a certain robustness to occlusion and change of viewpoint. Given two normalized histograms H_1 and H_2 with the same number of bins n_b, the Bhattacharyya coefficient $B(H_1, H_2)$ and the Histogram Intersection $I(H_1, H_2)$ are defined by:

$$B(H_1, H_2) = \sum_{i=1}^{n_b} \sqrt{H_1(i) \cdot H_2(i)} \tag{2.7}$$



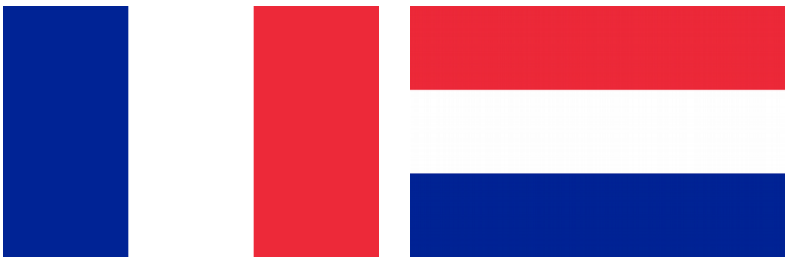
(A) Grayscale Image

(B) Grayscale histogram

FIGURE 2.3: Histogram taken from the first frame of *bolt* sequence [KPL+]

$$I(H_1, H_2) = \sum_{i=1}^{n_b} min(H_1(i), H_2(i)) \tag{2.8}$$

However, the main weakness of image histograms is directly induced by the used color space, and a non adapted color space may dramatically reduce the effectiveness of image histogram. Moreover, an histogram models an object globally, and while it indicates which colors are dominant or not, it does not indicate where they appear. Fig. 2.4 illustrates one example for which the Bhattacharyya coefficient will be 1.0, meaning that, for this measure, the two flags are identical, while they are visually (and semantically) different.



(A) French Flag

(B) Rotated version of the French flag

FIGURE 2.4: Two different flags with the same color distribution

This very low-level representation is, however, used for concrete applications: Comaniciu [CRM03] proposed the popular *Mean Shift* algorithm to track objects, using the classical RGB color space to create an object

model using the RGB histogram, and then track the target over one sequence using an approximation of the Bhattacharyya coefficient. Bradski extended the Mean shift, to propose CAMSHIFT [Bra98] for face tracking (tracking of face position, size and orientation) using HSV color space (he supposed that people's color hue is constant, contrary to color skin). Bradski's idea shows that, by selecting an adapted color space, some tasks can perform better (details in Section 3). In both cases, the use of color histogram, combined with light algorithms to exploit them, makes these trackers run above real-time. In the framework of our thesis, we set the limit of complexity of color-features at this level (even though, for detection purpose, we will even go lower, at the color pixel level).

2.1.1.3 Higher Level of representation

We saw on the previous part one very low level of color-based representation. In this section, we briefly mention some higher level color-based features used in computer vision.

Statistical features

Normalized color histograms can be interpreted in a probabilistic point of view: each bin represents the density of one color in the modeled image. Naturally, it is then possible to use some statistical measures, such as average, deviation, or moments. This higher level of representation has some advantages: given Eq. 2.1, in case of light intensity shift, or change ($a = b = c$ in Eq. 2.1), the standard deviation is invariant.

Mindru [Min+04] generalized the notion of moments at order $r = p+q$ and degree $d = a + b + c$ for a three-channeled image:

$$M_r^d(\mathbf{I}) = \sum_{x,y} x^p \cdot y^q \cdot I_{c_1}^a(x,y) \cdot I_{c_2}^b(x,y) \cdot I_{c_3}^c(x,y) \tag{2.9}$$

and used them to generate feature vectors from images, composed of moment invariants. His work aimed to match objects in case of illumination and viewpoint changes (tested on synthetic and real images). The method consists in generating several color moment invariants from moments computed by Eq. 2.9, and use them as basis of a vector space, on which images of objects are projected. Then, using MANOVA [JW+02], in the space of invariants, a subset optimizing a separation of different

classes of objects is generated. This subset then serves as a classifier for object matching.

Spatiogram

Color histogram lacks information about location of colors. Birchfield [BR05] proposed a model based on features called *spatiogram* for tracking context. His idea was to extend the histogram representation by considering for an image $\mathbf{I}$, and for each bin b, not only the number of pixels falling into b, but also the centroid and the covariance matrix of the spatial position of pixels falling into b. This representation gives an idea of how and where colors are spread. His representation also comes with a metric, to compare spatiograms of two objects. Fig. 2.4 illustrates two images with the same color histogram but different spatiograms (color centroids are spread horizontally for the first flag, vertically for the second).

SLIC superpixels

Superpixels were originally designed [RM03] for object detection. The principle was to group some pixels according to several criteria: texture, brightness, geometry. Each group of pixels, the superpixel, is then described and then used to train a two-class classifier. Several superpixel methods exist in the literature, and we refer to [Ach+12] for a coverage. We focus on one popular class of superpixels, proposed by Achanta [Ach+10], exclusively based on color features and pixels positions. In the paper, Achanta proposed a method to group pixels by, on the first hand, describing all pixels in a 5D space: three from the Lab color space, and the two pixels coordinates. On the second hand, he initializes superpixels' seeds by drawing them using a regular grid. Then, each pixel is associated to one seed according to a Euclidean-based distance (in the 5D space). SLIC superpixels have the advantage to require very low-level operations: Lab color space and Euclidean distance computations. Furthermore, in terms of usability, the only parameter to set is the number of seeds. While the main application of SLIC superpixels is segmentation, (and it has indeed been used for medical image segmentation), by describing each superpixels with multiple features, Achanta was able to perform object recognition. Fig. 2.5 illustrates SLIC superpixels computation.

Color Attribute

One computer vision task relying on color features is CBIR, for Content

(A) Input (B) Superpixels segmentation

FIGURE 2.5: Superpixel segmentation obtained from *ball* sequence [KPL+] with 400 superpixels.

Based Image Retrieval. Given a set of images, and a query ("Get all images containing red objects" for example), the goal is to get a subset of images matching the query. We focus on recent color-based feature developed by [VDW+09]. In his paper, Van de Weijer trains an algorithm to classify pixels according to color names. His training set is composed of annotated images (using the color of the object in the picture). Each image is, firstly, pre-processed (background removal and gamma correction), and then, represented into the Lab color space. Then, a color histogram is built for each image, and all histograms are sent to a classifier, trained to attribute a color label to each pixel. These models are called *color attribute*, and were used for color CBIR. Fig. 2.6 illustrates one mapping obtained using color attributes (with 11 colors). It has then been extended to tracking, action recognition and object detection [Kha+13]. This color feature, designed using annotated dataset, will be tested and evaluated for tracking in Section. 3.4. It is interesting to note that, computationally speaking, color attributes can be viewed as a low-level feature: the code provided by Van de Weijer outputs a LUT, mapping RGB pixels to color attributes. It is then as costly as moving from RGB to HSV color space for example, and can be computed on every pixel. But, from a conceptual point of view, we put color attributes on a higher level, as it is a result of a classification task, performed using annotated datasets.

Limits of color based features

As mentioned at the beginning of this section, color can describe specific objects, but not generic ones. As a consequence, it can be used for object tracking [CRM03]; [BR05] or for CBIR [VDW+09]. For tasks involving class of objects, such as object classification, color features

FIGURE 2.6: Color attribute of frame 0 of *marching* sequence

can not be directly used. In the literature, for this kind of task, we can however find color features, but used as complementary representation. We already mentioned Achanta' superpixels [Ach+10] who used SLIC for object recognition. We can also mention a representation method [VDSGS10], whose first step consists in projecting an image into different color spaces, then, for each channel, build a model of each image by using SIFT descriptors [Low99]. Two interesting points are important in this work: first, recognition accuracy depends on the chosen color space, and color spaces robust to intensity change or light color change are the most effective. Second, extracting other kind of features (in this case, geometrical features) from images projected into adapted color space increases the performance (the SIFT applied on grayscale images is outperformed by SIFT applied on different color spaces).

2.1.1.4 Conclusion

This section was dedicated on color-based features. To use color-based features in computer vision, the first choice is the color space. Many color spaces exist, but we describe three popular ones, that will be used and compared in the rest of this thesis: RGB, HSV and Lab.

The next step is the feature generation. One early feature is the color histogram. Easy to compute and lightweight, it is used in many tasks: tracking [CRM03], recognition [SB91], CBIR. From a statistical point of view, color-histogram can be extended to higher-level measures: average, variance... Birchfield [BR05] proposed an extension of color histogram for object tracking, by adding information about color location.

We also mentioned two other color-based features: the SLIC superpixels [Ach+10], originally designed for medical image segmentation, and used with other features for object recognition, and color attributes [VDW+09], for CBIR tasks. Even though color features are not directly

usable for object detection or recognition, they can be used in combination with other features for this task [Ach+10]; [Gal+11], or as a first level of representation [VDSGS10].

2.1.2 Shape-Based Representation

Previously, we presented some color-based visual features. Complementary way to visually describe objects is shape-based description. Extracting shape descriptors implies estimating geometrical quantities, such as gradient. Then, in this section, we will start by explaining methods to extend the notion of derivatives to images, and, as in the previous section, present some higher level geometrical features.

2.1.2.1 Mathematical context

The shape of an object is defined by its boundary, its form. In images, the shape is perceptible by the variations of pixels intensity. Mathematically, these variations can be measured by gradient, Hessian and other differential operators. In the next section, we will briefly present popular methods to approximate spatial derivatives on images by using convolutions. Then, in the next part, we will deal with blobs and junctions detection, based on Lindeberg theory [Lin98], and finally, present some higher-level features: sparse ones, such as keypoints and dense ones, such as the HOG [DT05] features.

2.1.2.2 Sobel filter

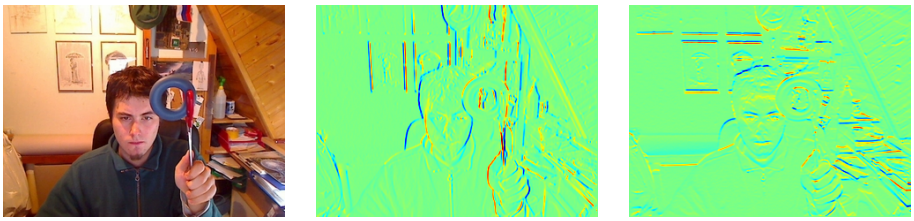
One very popular approximation of the gradient operator is the Sobel operator, defined by two kernels:

$$\begin{pmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{pmatrix} \tag{2.10}$$

$$\begin{pmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix} \tag{2.11}$$

which are convolved with the image to approximate the x- and y-derivatives. Fig. 2.7 illustrates Sobel filter applied to one image (blue stands for high

negative values, green for close to 0 values, red for high positive values). The main interest is the very low computation cost (the two filters are linearly separable). However, the Sobel operator is not robust against high-frequencies variation (noise for example).



(A) Input Image (B) Sobel in x-direction (C) Sobel in y-direction

FIGURE 2.7: Sobel in x and y applied in frame from *torus* sequence [KPL+]

Using these operators, by denoting $\mathbf{I}_x$ and $\mathbf{I}_y$ the gradient of $\mathbf{I}$ in direction x and y respectively, gradient magnitude $\mathbf{M}^{\mathbf{I}}$ and orientation $\mathbf{\Theta}^{\mathbf{I}}$ can be directly computed:

$$\mathbf{M}^{\mathbf{I}} = \sqrt{(\mathbf{I}_x)^2 + (\mathbf{I}_y)^2} \tag{2.12}$$

$$\mathbf{\Theta}^{\mathbf{I}} = \arctan(\frac{\mathbf{I}_y}{\mathbf{I}_x}) \tag{2.13}$$

Gradient magnitude and orientation are illustrated Fig. 2.8. These two equations will be essential for our work on tracking, as our goal is to propose a very light tracker based on low-level features. Gradient com-

(A) Input Image (B) Magnitude (C) Orientation

FIGURE 2.8: Magnitude and orientation from *torus* image. For a better visualization, we displayed only orientation for pixels with high gradient magnitude

putation with only Sobel filter is used in many algorithms: as we will see in Section 2.2.3, the Generalized Hough Transform, in its simplest form, only requires gradient evaluation, and combined with HSV color space,

can be directly used for an efficient object tracking algorithm [DG13]. For pedestrian detection, Tuzel [TPM08] trained a detector by building descriptors based on derivative features. Each region is described by first, computing 8 features images:

$$[x \quad y \quad |\mathbf{I}_x| \quad |\mathbf{I}_y| \quad \sqrt{\mathbf{I}_x^2 + \mathbf{I}_y^2} \quad |\mathbf{I}_{xx}^2| \quad |\mathbf{I}_{yy}^2| \quad \arctan(\frac{|\mathbf{I}_y|}{|\mathbf{I}_x|})] \tag{2.14}$$

and then computing the covariance matrix of these feature images ($\mathbf{I}_{xx}$ and $\mathbf{I}_{yy}$ are second order derivatives in direction x and y respectively). These features are then forming a Riemannian manifold serving as a space for classification. The trained classifier is based on boosting, which consists in training several weak classifiers (a weak classifier classifies correctly at least 50% of the training set), that are merged to a strong classifier, for a final classification.

Similarly, it is possible to define other differential operators with convolution operations: the 4-connected Laplacian operator has been much used for edge detection, and can be defined with this kernel:

$$\begin{pmatrix} 0 & +1 & 0 \\ +1 & -4 & +1 \\ 0 & +1 & 0 \end{pmatrix} \tag{2.15}$$

2.1.2.3 Local Jet space

In the last section, we presented the Sobel filter, to approximate the notion of derivative in image processing. This section will be dedicated to a formulation of derivative that we will use in our work.

Using distribution theories, it is possible to define a notion of gradient and Hessian not directly with an image $\mathbf{I}$, but with the convolved image $(\mathbf{I}*g)$, with g a function, called *smoothing* function, differentiable as many times as necessary. Indeed, in this case, the derivative in the direction i, denoted $\frac{\partial \bullet}{\partial i}$, is then defined by (according to derivation in the distribution space):

$$\frac{\partial(\mathbf{I} * g)}{\partial i} = \mathbf{I} * \frac{\partial g}{\partial i} \tag{2.16}$$

The Sobel operator represents a certain trade-off between approximation of the gradient, and speed. However, as we will see in this part, the formulation given by Eq. 2.16 can provide a richer representation.

Definition

In 1983, Witkin submitted a patent [Wit87] entitled "Scale-space filtering" dealing with the detection of some important points in a signal, such as maxima or discontinuities. The key point was based on Eq. 2.16, where g is a function with two parameters: x, a spatial coordinate, and σ, a scale, such that:

$$g(x,\sigma) = \frac{1}{\sigma \cdot \sqrt{2 \cdot \pi}} \cdot e^{-\frac{x^2}{2 \cdot \sigma^2}} \tag{2.17}$$

This scale term σ provides a method to adjust the "smoothing effect": the higher σ is, the more $\frac{\partial(\mathbf{I}*g)}{\partial i}(x)$ will be influenced by pixels around x.

For the rest of this thesis, let $\mathbf{I}^{\sigma}_{x^i y^j}$ be the Gaussian derivative of $\mathbf{I}$ at order i at direction x and j at direction y, and with a scale σ. When the derivative order is 0, we will omit the direction in the notation ($\mathbf{I}^{\sigma}_{x^i y^0} = \mathbf{I}^{\sigma}_{x^i}$)

This formulation was at the basis of the now called *scale-space theory*. Koenderink [KVD90] proposed a family of receptive fields based on Gaussian derivatives in different directions in order to detect edges.

Working in the space of derivatives at several orders, and with a scaling function has interesting properties. For a given image $\mathbf{I}$, the *local jet space* at order n, and scales $\{\sigma_1, \sigma_2, \ldots, \sigma_p\}$, is defined by the projection of $\mathbf{I}$ into the space $\mathbf{LJ}(I) = \{\mathbf{I}^{\sigma}_{x^i y^j}\}_{0 \leq (i+j) \leq n,\, \sigma \in \{\sigma_1, \sigma_2, \ldots, \sigma_p\}}$. This space has interesting properties, and this level of representation is already sufficient for some high-level tasks. The term *local jet* is due to two points:

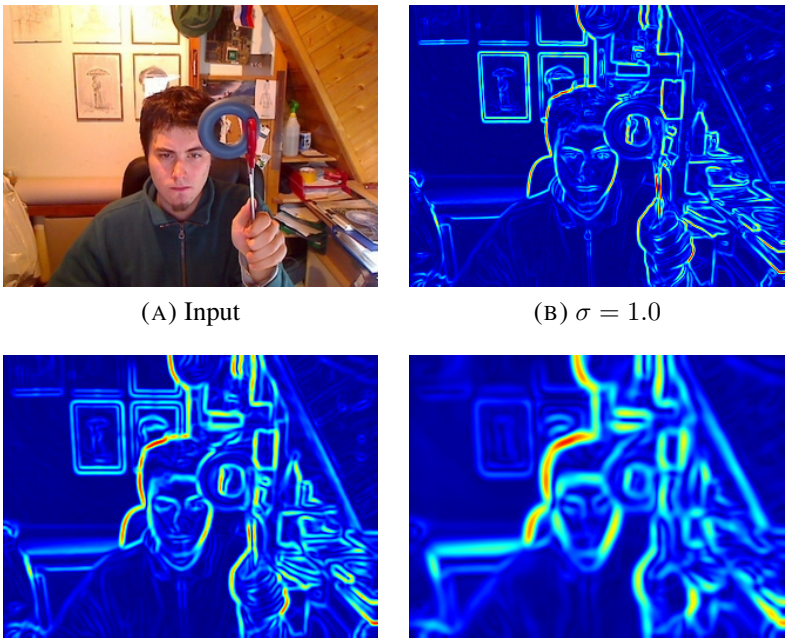
1. In mathematics, a jet at order n is an operator taking as an input a differentiable function f, and giving as an output the Taylor formula at order n

2. The notion of locality is related to the scale σ of the smoothing function

In the whole thesis, $\mathbf{I}^{\sigma}_{x^i y^j}$ will be computed by convolving $\mathbf{I}$ with the kernel defined by:

$$G^{\sigma}_{x^i} \cdot G^{\sigma}_{y^j} \tag{2.18}$$

where $G^{\sigma}_{x^i}$ (respectively $G^{\sigma}_{y^j}$) is a square Gaussian matrix whose component at direction x (respectively y is defined by the i-th derivative of the Gaussian function of variance σ^2, and size $2 \cdot \sigma^2 + 1$. The kernel obtained with Eq. 2.18 is then normalized in order to have a maximum at 1. Fig. 2.9 illustrates the impact of σ for gradient calculation.



(A) Input (B) $\sigma = 1.0$

(C) $\sigma = 2.0$ (D) $\sigma = 4.0$

FIGURE 2.9: Gradient magnitude at different scales

Visual features extraction

From the derivatives, it is possible to detect interest points, at different scales, such as blobs and junctions. Lindeberg [Lin98] proposed a method to evaluate the *blob-ness* $\mathcal{B}$ and the *junction-ness* $\mathcal{J}$ of a pixel p, at scale σ:

$$\mathcal{B}(\mathbf{I}) = \sigma \cdot \det(\mathcal{H}^{\sigma}(\mathbf{I})) \tag{2.19}$$

$$\mathcal{J}(\mathbf{I}) = (\mathbf{I}_y^{\sigma})^2 \cdot \mathbf{I}_{x^2}^{\sigma} - 2 \cdot \mathbf{I}_x^{\sigma} \cdot \mathbf{I}_y^{\sigma} \cdot \mathbf{I}_{xy}^{\sigma} + (\mathbf{I}_x^{\sigma})^2 \cdot \mathbf{I}_{y^2}^{\sigma} \tag{2.20}$$

with $\mathcal{H}^{\sigma}$ being the Hessian operator computed at the scale σ, $\det$ the determinant operator. Fig. 2.10 illustrates location of highest junction-ness and blob-ness values for one frame taken from *singer2* sequence [Kri+15]. We computed both measures at different scales (1, 2, 4) and at each scale, search for the 10 highest peaks. Highest peaks are in red, lowest in blue. Crosses and circles sizes correspond to the scales.

FIGURE 2.10: Junctions and blobs at different scales

Another relevant measure in the 2D jet space was proposed by Griffin $\mathcal{G}_\sigma$ [Gri06]:

$$\begin{aligned}\mathcal{G}_\sigma(\mathbf{I})^2 = \sigma^2 \cdot ((\mathbf{I}_x^\sigma)^2 + (\mathbf{I}_y^\sigma)^2) + \\ \frac{1}{4} \cdot \sigma^4 \cdot (\mathbf{I}_{x^2}^\sigma + \mathbf{I}_{y^2}^\sigma)^2 + \\ \frac{1}{4} \cdot \sigma^4 \cdot ((\mathbf{I}_{x^2}^\sigma - \mathbf{I}_{y^2}^\sigma)^2 + 4 \cdot (\mathbf{I}_{xy}^\sigma)^2)\end{aligned} \tag{2.21}$$

Fig. 2.11 illustrates Griffin norm computed at different scales (red stands for high values). It is interesting to note that maxima are located at different places, from one scale to another.

Crosier [CG10] classified pixels according to their 2nd order jet into different class of structures: flat, slope, saddle point, local maxima or ridge. To do that, given a scale σ, he first defines normalized derivatives at every order (i, j): $s_{i,j,\sigma}^{\mathbf{I}} = \sigma^{i+j} \cdot \mathbf{I}_{x^i,y^j}^\sigma$. He then computes different terms:

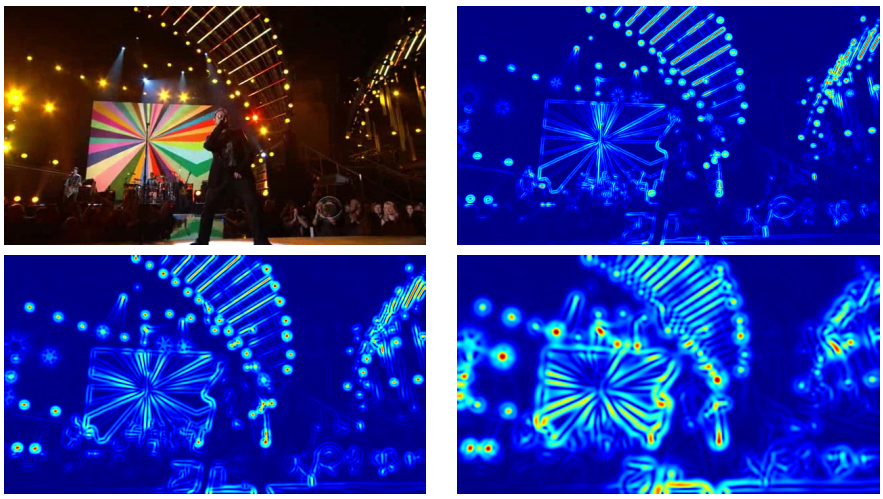


FIGURE 2.11: Griffin norm computed at scales $\{1, 2, 4\}$

- The zero-order term: $\epsilon \cdot s^{\mathbf{I}}_{0,\,0,\sigma}$ (ϵ will be defined later)
- The gradient magnitude term: $2 \cdot \sqrt{\left(s^{\mathbf{I}}_{1,\,0,\sigma}\right)^2 + \left(s^{\mathbf{I}}_{0,\,1,\sigma}\right)^2}$
- The trace of the Hessian matrix: $\lambda = s^{\mathbf{I}}_{2,\,0,\sigma} + s^{\mathbf{I}}_{0,\,2,\sigma}$
- The residual term: $\gamma = \sqrt{(s^{\mathbf{I}}_{2,\,0,\sigma} - s^{\mathbf{I}}_{0,\,2,\sigma})^2 + 4 \cdot \left(s^{\mathbf{I}}_{\sigma 1,\,1)}\right)^2}$
- And two final terms, linked to the eigenvalues of the Hessian matrix: $\frac{\lambda+\gamma}{\sqrt{2}}$ and $\frac{\lambda-\gamma}{\sqrt{2}}$

and then classifies pixels, at this given scale, according to the highest value from those indicated Tab. 2.1. ϵ is chosen to adjust the amount of "flat" pixels we want to have. The sign is linked to the polarity of the pixel, and working with absolute values and neglecting polarity is also possible. Then, after describing each pixel from a texture image with this classification and by using different scales, Crosier built a global histogram from this image and used it for texture classification.

Highest value	Class
$\epsilon \cdot s^{\mathbf{I}}_{0,\,0,\sigma}$	Flat
$2 \cdot \sqrt{\left(s^{\mathbf{I}}_{1,\,0,\sigma}\right)^2 + \left(s^{\mathbf{I}}_{0,\,1,\sigma}\right)^2}$	Slope
$+\lambda$	Local maxima
$-\lambda$	Local minima
γ	Saddle point
$\frac{\gamma-\lambda}{\sqrt{2}}$	Positive ridge
$\frac{\gamma+\lambda}{\sqrt{2}}$	Negative ridge

TABLE 2.1: Crosier's classification, at a given scale σ

Applications

In terms of application, this level of representation has already proven usable in diverse applications: Manzanera [Man11]; [Man10], using many scales at the same time, applied this generated representation for image denoising, optical flow estimation, background subtraction.

For denoising case, Manzanera [Man11] adapted the NL-means denoising algorithm [BCM05]. In Buades paper, the denoising function $\mathbf{ML}(p)$ is performed at pixel p, and its neighbourhood $N(p)$ by:

$$\mathbf{ML}(p) = \frac{1}{Z(p)} \sum_{x \in N(p)} \mathbf{I}(x)\omega(p, x) \tag{2.22}$$

where ω is function of the difference of intensity between $\mathbf{I}(x)$ and $\mathbf{I}(p)$. Manzanera [Man10] enhancement of the NL-means is to change the function ω, from one defined by the sum of squared differences between two image patches centered on x and p respectively, to another using the distance between the projections of x and p in the local jet space.

In [Man11], Manzanera used the local jet space for two tasks:

- For optical flow estimation: given a sequence, at a frame $t + 1$, the optical flow of a pixel p is estimated by looking for the nearest neighbor of $\hat{p}$, projection of p in the local jet space and then finding the corresponding pixel in the image space

- For background subtraction: the algorithm is an adaptation of ViBe background subtractor [BVD11]. The principle of ViBe algorithm is to model the background by a sample $\mathcal{P}$ of pixel values projected into a color space. Then, at the frame $\mathbf{I}_t$, one pixel p is considered as background if, in the selected color space, the sphere of radius R (defined by the user) and centered in p contains a number of pixels of $\mathcal{P}$ above a threshold (also to define). Manzanera's background subtractor relies on this principle, by substituting to the color space the local jet space

We defined different formulations of differential operator in this section. We can find in the literature different works using this level of shape-features. Similarly, we aim to work exclusively on these features.

2.1.2.4 Higher level features

From the lowest level geometrical feature based on gradient and Hessian, it is possible to extract some higher geometrical features. These features are diverse, and in this part, we will briefly present two different families of features.

Keypoints

The first type is the keypoint-based features. We will present two levels of features: the keypoint detection, and a higher one, the keypoint descriptor.

Given an image $\mathbf{I}$, extracting a keypoint means localizing it. This keypoint is usually a point where shape is particular, such as corners. In computer vision, one of the early keypoint detectors is the corner detector of Harris [HS88], which requires only the first derivative operator. Indeed, after evaluating derivatives at direction x and y, for all pixels $p_0 \in \mathbf{I}$, Harris defines a neighborhood $N(p_0)$, and generates a matrix $\mathbf{M}(p_0)$ such that:

$$\mathbf{M}(p_0) = \sum_{p \in N(p_0)} \begin{pmatrix} \mathbf{I}_x^2(p) & \mathbf{I}_x(p) \cdot \mathbf{I}_y(p) \\ \mathbf{I}_x(p) \cdot \mathbf{I}_y(p) & \mathbf{I}_y^2(p) \end{pmatrix} \tag{2.23}$$

Then, Harris uses a measure of interest of a pixel p_0: $R(p_0) = \det(M(p_0)) - k \cdot \mathrm{tr}(M(p_0))^2$, where $\det$ is the determinant operator, tr the trace operator, k a constant. Then, a threshold operation, combined with a non-maxima suppression is used to extract keypoints. One main interest of Harris keypoint is its robustness to orientation and scaling change. However, weakness of keypoint-based features is the sparsity of the descriptor: in case of few detected keypoints, object description may not be representative. Even though the threshold operator allows to adjust the number of detected keypoints, those having low interest value may not be reliable. Fig. 2.12 illustrates Harris corner detector. This detector can be time consuming for critical cases.

Rosten [RD06] proposed a very fast method to detect corners, based on a geometry of a corner. His first idea consists in detecting keypoints from an image, by considering that p is a keypoint if, given a circle of center p and circumference 16 pixels, there exists a contiguous set of n pixels brighter or darker than p. This idea is close to the visual perception of corner: an area composed of two regions of different intensities, whose frontiers form an acute angle. This method, called FAST algorithm, illustrated Fig. 2.13, and with optimization detailed in [RD06], is used to train

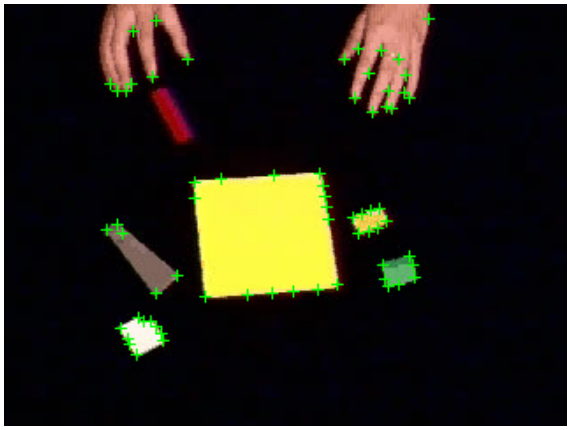


FIGURE 2.12: Corners detected in *lego1* sequence

a decision tree classifier as a corner detector.

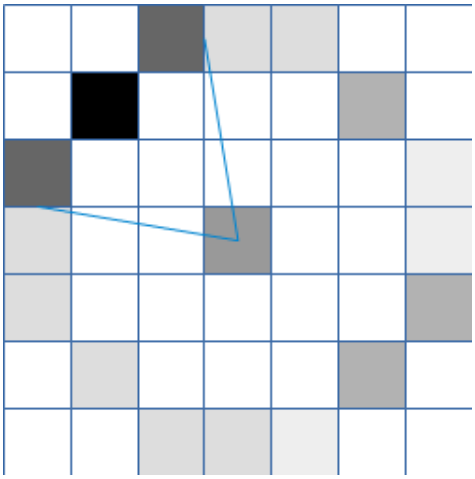


FIGURE 2.13: Fast keypoint schematic.
The central point is detected as a keypoint by the FAST algorithm: in the circle of circumference 16, there is one large contiguous set of pixels brighter than the central point.

Corner detection, combined with an optical flow estimator such as Lucas-Kanade method [LK+81], allows to estimate some geometrical transformation of objects: by modeling this object with a set of keypoints and estimate their inter-frame trajectories, geometrical transformation can be estimated. For object tracking context, some effective trackers [KMM10] also relies on keypoint-based representation and Lucas-Kanade method (this will be detailed further, at Section. 3).

Harris method only localize keypoints. Another popular keypoint-based feature was proposed by Lowe [Low99] in the now-called SIFT keypoints. Lowe starts by building a multi-scale space, by convolving an

image $\mathbf{I}$ with Gaussian at different scales, and computing the difference between two convolved images at adjacent scales. Then, the localization step done by searching local maxima, and a first level of description (localization, scale and orientation) is applied to the detected keypoints. The second contribution of [Low99] is the SIFT descriptor: each detected keypoint is represented by histograms of gradient orientation from areas surrounding it. SIFT keypoint is a very popular keypoint-detector and descriptor (more than 12 000 citations) and led to different alternatives: SURF [Bay+08], FREAK [AOV12] are among popular keypoints. In terms of application, SIFT has proven to be a versatile feature: originally designed as an object recognition algorithm, it has proven to be effective in tracking context [MP13] and detection [LLS08].

Histogram of Oriented Gradient (HOG)

The last shape-based feature that we would like to present, and that we will use for detection task is the HOG (for Histograms of Oriented Gradients) features [DT05], originally used for Human Detection, and providing a dense level of representation. The principle consists in subdividing an image into multiple (rectangular or circular) cells. In each cell, a gradient orientation histogram is computed, each pixel's contribution being proportional to its gradient magnitude. To be robust to local illumination variation, gradients are normalized: several cells are grouped into blocks, in which gradient magnitude are normalized. Finally, the concatenation of all these histograms is the so-called HOG descriptor. For human detection, Dalal trains a SVM classifier as a detector. [DT05] not only presents the descriptors and results, but also provides a full study a HOG features: shape of cells (rectangles or circles), number of bins, use or not of signed orientations, methods to normalize the gradient magnitude... When proposed by Dalal and Triggs for human detection [DT05], HOG features proved to be competitive, as it outperformed state-of-the-art methods. Others detectors are also based on HOG features, and notably [Gal+11] that we will detail further in Section. 4. In tracking context, we can mention [Dan+14a] who proposed an accurate correlation-based tracker using HOG features.

2.1.2.5 Conclusion

In this part, we focused on shape-based features. All shape-based features are based on differentials. We started to define the Sobel filter to compute gradient. Then, the formulation defined by Eq. 2.16 leads to scale-space theory [Lin98], essential for our work. We will exploit this for tracking and detection context. We also presented some higher level features extracted from low-level shape features. The first class was the keypoint features: Harris corner detector [HS88], SIFT [Low99] and FAST [RD06]. Another feature we mentioned, and that will be used in Section. 4 was the HOG feature [DT05]. Unlike keypoint-based ones, they can provide a dense description of images (available at each pixel).

2.1.3 Conclusion of the section

We made a review of visual-based features, including color-based and shape-based ones. In both cases, the first step was to explain how to represent the visual feature: choosing the color space on the one hand, computing derivative on the second hand. Then, we detailed some low-level features built from them, color histogram and gradient orientation. We aim to work exclusively on color histograms and derivatives in the whole thesis. From these low-level features, we also mentioned some high-level features, directly derived from low-level ones: statistics or SLIC superpixels for the color part, keypoints or HOG for shape-based part. For our work, we are willing to work on local low-level features: color pixels for the color part (aggregated for color histogram in tracking context), scaled derivatives for the other. Fast to compute but with lower power of representation than higher features, we aim to study their benefits and limits in tracking and detection contexts.

Other classes of features exist, such as texture-based features, LBP, or filter-banks features, such as wavelet-based (for example Haar features [Haa10]). We also choose not to deal with temporal features, such as motion-based one, only usable in videos. More recently, using deep learning autoencoders, several pre-trained features are available to the community: MatConvnet [VL15], AlexNet [KSH12]. Even though these features have outranked most of the traditional features in several tasks (especially in image recognition, and see Section. 3.2 for tracking case), until recently, they did not prove efficient in terms of speed (due to their time consumption, most of them require GPUs), and are not as flexible as low-level

features. Fig. 2.14 shows all mentioned features in this part. In dashed line, the framework we are working on.

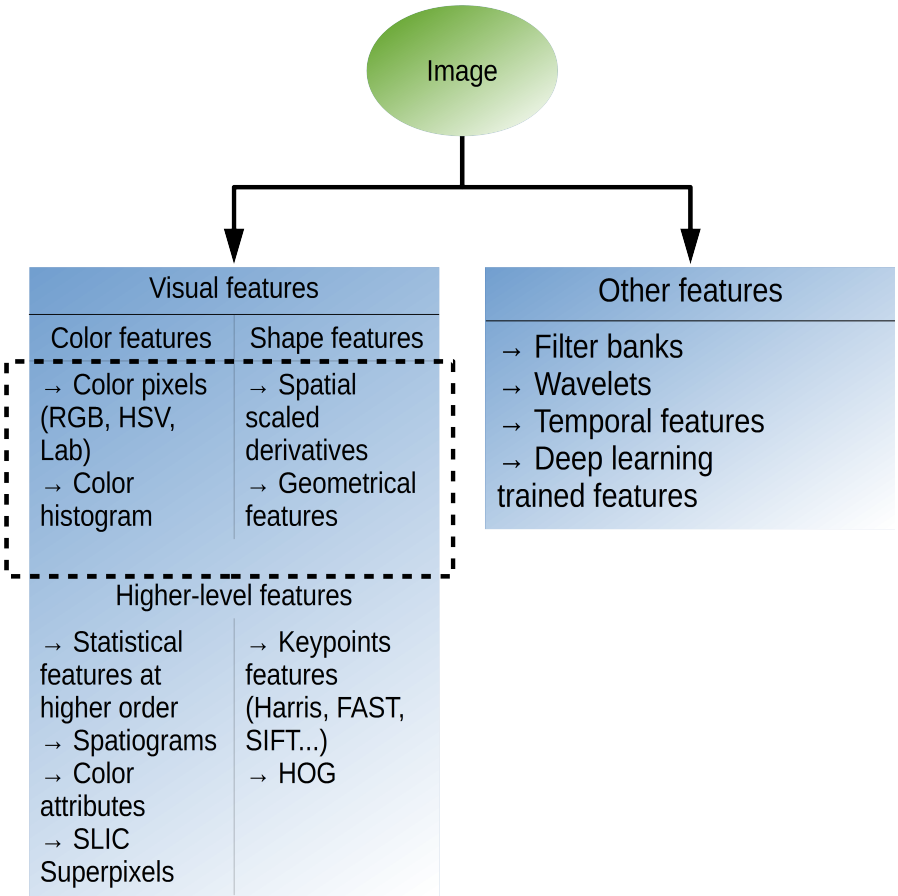


FIGURE 2.14: Summary of different presented features

The next section will be fully dedicated to the Hough Transform, the main spatial aggregation mechanism of our work.

2.2 Hough Transform

In this section, we will focus on methods to exploit visual features, in order to extract information. Many algorithms have been designed in order to do this. Most of the time, these algorithms are task-dependent and can come from diverse areas: statistics [IB98]; [CRM03], Machine Learning [Gal+11]; [LLS08], signal processing [Dan+14b]... In our work, we choose to focus on one algorithm: the *Hough Transform* [Hou62].

This algorithm was originally proposed by Hough in a patent for tracking particles in bubble machines. From the formalization made by Duda [DH72], followed by the generalization of Ballard [Bal81] to nowadays,

this algorithm became one of the most famous in computer vision. The Hough Transform has been adapted to many tasks. This section will be organized that way:

- First, we will focus on the history of the Hough Transform
- Second, we will formally define it, and present an overview of different Hough Transforms
- Third, we will define the Generalized Hough Transform (GHT), that has a central role in our work
- Fourth, to insist on the versatility of the Hough Transform, we will present some real-life applications performed using the HT

2.2.1 History of the Hough Transform

Considering its importance in our work, we now present the history of the HT [Hou62].

Historically, Paul V. C. Hough submitted a patent [Hou62] on the detection of lines in pictures. It was especially designed to track trajectories of particles in bubble chambers. While the main part of the patent is dedicated to the description of the whole electronic system, with its different components, the main idea of the now-called Hough Transform is present (see Fig. 2.15). Indeed, for a given line (102, 104, 106 in Fig. 2.15), all points (in the image space) from these lines can be transformed into a specific line (lower image of Fig. 2.15) in the parameter space (the now called *Hough space*). Then, for a set of colinear points, all the lines obtained by this particular transformation are intersecting at a particular point, characterizing the line linking all points. Even though no mathematical model was designed, the idea to make one image element (in this case, a point) vote into a parameter space (one line in the lower part of Fig 2.15) was present. Then, according to Hart [Har09], Rosenfeld [Ros69] was the first to propose a mathematical form to this transformation. Given a point (x_0, y_0), the transformation associates this point to the line:

$$y = y_0 \cdot x + x_0 \tag{2.24}$$

Every couple (x, y) verifying this equation is a couple of parameters of one line passing through (x_0, y_0) in the image space. Reciprocally, for a set of aligned points (x_i, y_i) in the image space, all lines obtained by

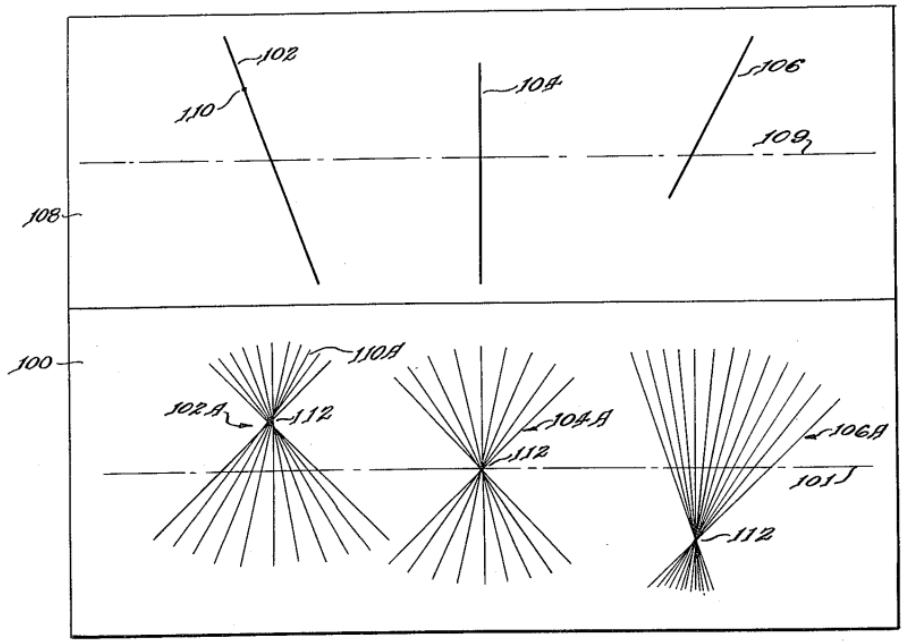


FIGURE 2.15: Figure from Hough's patent. Each point from lines of the upper part generates one line in the lower part. Then, all aligned points from the upper part generates a beam of concurrent lines in the lower one

this transformation will intersect at a certain point of the parameter space, defining line's parameters to detect. However, for lines parallel to the x-axis (all the y_i are equals), this formulation does not work (all the lines obtained by transformation are parallel), and Rosenfeld suggests to swap x_i and y_i to solve this issue.

2.2.2 General formulation

A decade later, Duda and Hart [DH72] finally gave the formulation which is still in use today. For lines detection, they choose a polar parametrization (r, θ) (see Fig. 2.16): r is the length of the vector $\overrightarrow{OP}$, with P being the orthogonal projection of the origin O to the line, and θ ($\theta \in [0, \pi[$), the angle formed by the x-axis with $\overrightarrow{OP}$, and so, a line (r, θ) is defined by the set of pixels (x, y) such that:

$$r = x \cdot \cos\theta + y \cdot \sin\theta \tag{2.25}$$

With this transform, a single point is then transformed into a sinusoid curve. Detecting lines consists then in detecting couples (r, θ) in Hough

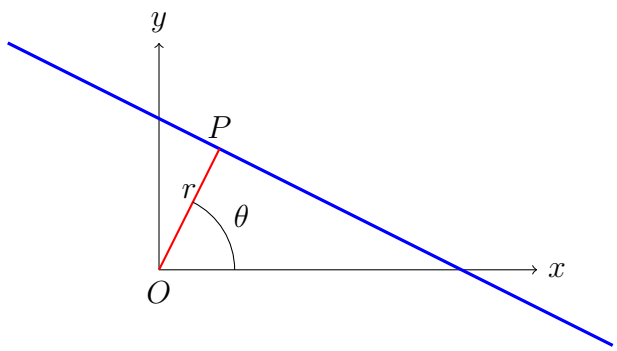


FIGURE 2.16: Parametrization of the blue line using Duda [DH72] parameter set (r, θ)

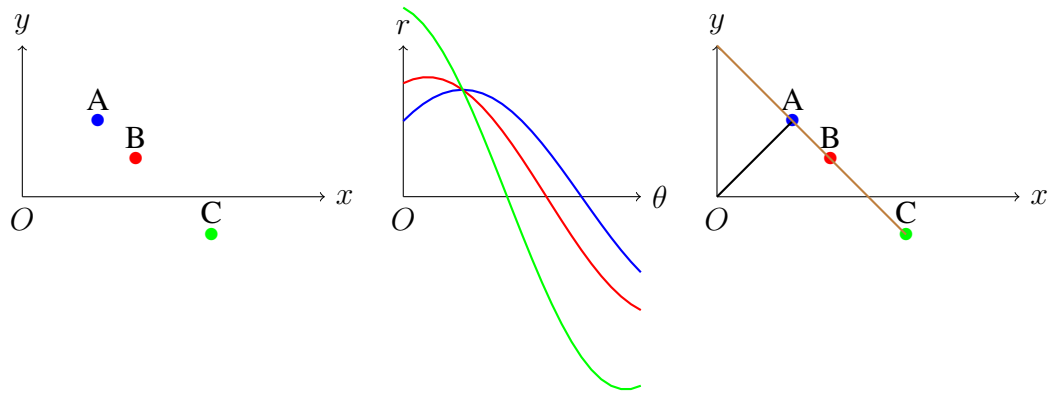
space where many sinusoids pass (see Fig. 2.17). This principle is important for the Hough Transform: the more there are sinusoids passing through (r_0, θ_0), the more likely there is a line of parameters (r_0, θ_0). Thanks to this property, the Hough Transform is partially robust to occlusion and noise. In practice, the principle is to define a threshold value τ such that, each couple (r, θ) which has accumulated more than τ votes (or for which more than τ sinusoids are intersecting) corresponds to a line. Moreover, thanks to the hypothesis $\theta \in [0, \pi[$, one line is uniquely defined by the couple (r, θ). And, unlike Rosenfeld parameter space, this one is suitable for every kind of lines, even those parallel to the x-axis.

In their article, Duda and Hart [DH72] also showed how to detect other parametric curves, such as circles. The principle is similar to the line detection: given a circle defined by its center (x_c, y_c) and its radius r, then with equation:

$$(x - x_c)^2 + (y - y_c)^2 = r^2 \tag{2.26}$$

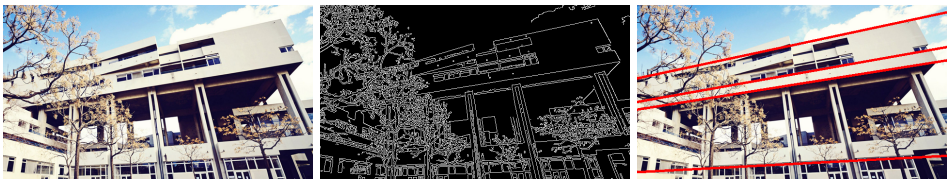
The Hough Transform can detect circles by considering a set of pixels, and make each pixel vote in the 3D parameter space (x_c, y_c, r). Illustrations of the line and circle Hough Transforms can be seen in Fig. 2.18 and Fig. 2.19. More generally, given a shape defined by an equation, the Hough Transform is able to detect this shape by taking an image, binarize it (with an edge detector for example), and make all positive pixels vote into the multidimensional parameter space.

The main drawback of the HT is the time and memory consumption. The higher the dimension is, the greater the computation time and the memory consumption are. For example, let us consider the case of circle detection in an image of resolution 500×500. Let us also suppose that the Hough space is defined by three parameters: the abscissa and ordinate



(A) Points (B) Hough Transform (C) Line linking A, B and C

FIGURE 2.17: Line Hough Transform on three points: $A(1,1)$, $B = (1.5, 0.5)$ and $C = (2.5, -0.5)$



(A) Input Image (B) Edges (Canny-Deriche filter [Der87]) (C) Strongest lines obtained by Hough Transform

FIGURE 2.18: Lines detection using Hough Transform

(A) Input Image (B) Edges (Canny-Deriche filter [Der87]) (C) Strongest circles obtained by Hough Transform

FIGURE 2.19: Circles detection

of the center and the circle radius. We can suppose that the circle center is inside the image, and the circle radius can vary from 1 to 500 pixels. In this case, the Hough space has a size of about 125 Mo. If we focus on computation time: each voting pixel p will vote, at each radius r, for a circle of center p, and radius r. Moreover, after the vote process, the peak detection will take place in the whole Hough space. Both operations are computationally expensive for circle detection, and this problem is even more critical for more complex shapes (ellipses for example). One

solution can be to quantize the Hough space to reduce the memory footprint and to accelerate the Hough voting. However, the accuracy of the estimation is impacted.

2.2.2.1 Variant of the Hough Transform

Hough Transform is a very popular algorithm in computer vision, and many surveys [IK88]; [MC15] provide a summary of its different variants. In this section, we will present some Hough-based algorithms, and explain their advantages and disadvantages compared to the original version [DH72]. However, due to its important role in our work, we will develop the Generalized Hough Transform [Bal81] only in the next section.

The first enhancement we will detail is the Probabilistic Hough Transform [KEB91]. The improvement is done in terms of computation time: while the original one makes vote all N extracted elements (pixels contour), the Probabilistic Hough Transform randomly draws n voting elements (with $n << N$). The aim of the method is to find a trade-off between speed and accuracy of the shape detection: the lower n is, the faster the algorithm it is, but lesser its accuracy is. This method has two weaknesses: noisy elements among the N ones (inducing wrong votes) and short lines detection (low probability to draw elements from it).

The second variant of the Hough Transform we would like to mention is the Randomized Hough Transform [XO09]; [XOK90]. For line detection, it relies on the principle that one unique line passes through a couple of points. The principle is then to make a "many-to-one" vote (opposed by the "one-to-many" vote of the Hough Transform, where one pixel votes for a set of parameters), by randomly drawing two elements, and make the formed couple vote for one unique couple of parameters. In this condition, the Hough space can then be modelled by a simple codebook indexed by the two parameters (r, θ), compared to the original Hough Transform, which contains the whole Hough Transform map. Then, by storing only couples (r, θ) that received a vote, the memory consumed by the Randomized Hough Transform is lower than the original one. Detecting lines then consists in selecting entries of the list with an accumulation value above a certain threshold.

Both HT variants are less robust against noise than the original HT. Kiryati [KKA00] compared these two variations by considering synthetic

images of p points along a line, and n noisy points. With these images, the first goal was to estimate efficiency of the two algorithms, and second, to estimate the robustness to noise (by increasing n). It has been shown experimentally that for the same accuracy, the Randomized Hough Transform is faster than the Probabilistic one. However, for very noisy images, its accuracy is much lower.

These two algorithms are popular variants of the Hough Transform. The list is not exhaustive, and we will refer to [MC15] for a coverage. In the next section, we will focus on the Generalized Hough Transform, the algorithm that will be the core of our thesis.

2.2.3 Generalized Hough Transform

In 1981, Ballard proposed the Generalized Hough Transform [Bal81], designed to build a model of any kind of shapes (in particular those that can not be modeled by a parametric equation), and then detect them in a query image.

First, it requires to create the *R-Table* denoted R, used to represent the object. Given an image $\mathbf{I}$ and an object O, let C be the contour of O (generated by edge detection algorithm for example), and r a reference point (for example the object center). For all pixels $p \in C$, let θ_p be the (quantized) orientation of its gradient. Then, we store in $R(\theta_p)$ the vector $\vec{u} = \overrightarrow{pr}$:

$$R(\theta_p) = \{\vec{u} | \exists p \in C : \Theta(p) = \theta_p, \vec{u} = \overrightarrow{pr}\} \tag{2.27}$$

with Θ the orientation map. Fig. 2.20 illustrates the construction of the R-Table.

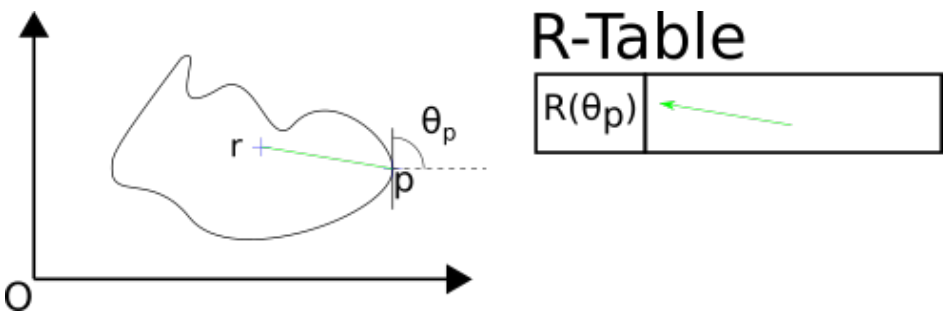


FIGURE 2.20: Building the R-Table. On the left part, a contour image, with a reference point r. For each pixel p from the contour, the gradient orientation θ_p is computed, and contributes to the R-Table in the entry $R(\theta_p)$, by the displacement $\overrightarrow{pr}$

Second, for shape detection (the Generalized Hough Transform itself), given an image test $\mathbf{T}$ and its contour map C, the aim is to build an accumulation map $\mathbf{HT}$. For all pixels x, $\mathbf{HT}(x)$ is equal to the number of pixels $p \in C$ which have voted for x:

$$\mathbf{HT}(x) = |\{p \in C | \exists \vec{u} \in R(\theta_p), x = p + \vec{u}\}| \tag{2.28}$$

with $|C|$ the cardinality of the set C. Then, a peak detection in $\mathbf{HT}$ provides possible instances of the object O, or more precisely, possible locations of the reference point r. One simple solution of peak localization can be a simple argmax operation:

$$p_{peak} = \underset{p}{\mathrm{argmax}}(\mathbf{HT}(p)) \tag{2.29}$$

Fig. 2.21 illustrates the GHT for a synthetic example, with the shape shown Fig. 2.20. From the contour image on the left, the GHT is computed and image on the right is obtained: the bigger and the redder the peak is, the more likely it is to find the shape center. Fig. 2.22 illustrates a GHT resulting from a R-Table built using the sheep inside the blue rectangle. The GHT is shown on the right, and the red peak corresponds to the prototype sheep. The GHT can be seen as an extension of the Hough

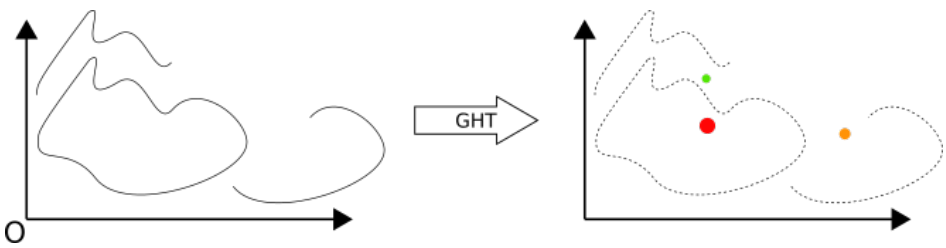


FIGURE 2.21: Generalized Hough Transform on the contour image (on the left), and the Hough Transform on the right

Transform in the sense that, given an image, it extracts some elements, makes them vote into a parametric space (the accumulation map $\mathbf{HT}$), and then detects the presence of the shape with a peak analysis. The difference with the standard Hough Transform is the presence of the *R-Table*, modeling any shape.

This R-Table embodies the versatility of the GHT (see Section. 2.2.4), by providing the possibility to consider higher level features. If we want

to stay in the domain of shape-based features and exclusively on derivative levels, [Tsa97] proposed to project the pixel contour of the shape not only in the gradient orientation space, but in the 3D space composed by gradient orientation, a concavity measure and the radius of the curvature. The GHT is then processed in two times: first to estimate the target orientation, then to estimate object centroid. In tracking and detection context, we will see how the GHT can be extended with even higher level features.

FIGURE 2.22: GHT on a scene, with the squared sheep used as a model

The GHT presents a certain robustness to partial occlusion or small deformations: considering that the Hough Transform in an image $\mathbf{I}$ is equal to the sum of the partial Hough Transforms of n disjoint subsets $\mathbf{I}_i$, the GHT computed to detect a shape in occlusion of small deformation case can be identified to the GHT realized on $\mathbf{I}$ excluding some of its subsets. Regarding the parameter space, the more there are parameters, the higher the memory footprint is, and the higher the time consumption is. For example, Ballard proposed, to handle scale and rotation variation, not only to vote in the spatial space, but also for different scales and orientations. In this case, if we define as d the size of $\mathbf{HT}$, s the number of scales and o the number of orientations, the multi-scale and multi-orientation GHT requires a memory footprint of $d \cdot s \cdot o$, and each pixel will also vote $s \cdot o$ more times.

Conceptually, the Generalized Hough Transform consists in using a codebook (the index of the R-Table) to model and detect any kind of shape. As we will see in the two next chapters, this codebook, can be

composed of different features: orientation for the original GHT, key-points features, color-based ones, and can also have diverse forms: arrays, decision trees. This is a major element of the versatility of the GHT. To corroborate this, in the final part of the section, we will present different tasks involving the Hough Transform.

2.2.4 Applications of Hough Transforms in computer vision

Hough Transform is one of the most famous algorithms in computer vision: Duda's paper [DH72] was cited more than 5 500 times, while Ballard's one [Bal81] was cited more than 4 400 times. From a line detection, it has firstly been extended to any shape modeled by an equation, and then generalized to any kind of shape. Currently, Hough Transform has been applied for different tasks, in different areas. In this section, we will see some applications of the Hough Transform. For tracking and detection cases, we will refer to Section. 3 and Section. 4 respectively.

- Hough Transform have been designed for line detection. This task, can be used for many applications: lane detection [Sat+10], robot navigation [FLW95]...

- In biometry, Hough Transform can be used for different modalities: iris localization [TBA02] or segmentation [Tia+04], or in fingerprint matching [PFJ13], where is was used to align two fingerprints: the minutiae are extracted and described using a specific descriptor [CFM10], the Hough Transform is then applied by extracting pairs of minutiae (one for each fingerprint), and make them vote for a geometrical transformation that matches them

- For medical imaging, Hough Transform can be used to detect different organs, obtained from diverse sensors. For ventricle myocardum localization, [MSN12] applied the circle Hough Transform at every layer of image from a 3D cardiogram. Arterial diameter estimation is also done using circle Hough Transform on ultrasound images [Gol+06]. The Generalized Hough Transform has also been used for 3D segmentation of the heart, from Computer Tomography images [Eca+08]. The novelty proposed in [Eca+08] was to integrate several reference shapes in one unique R-Table

- Hough Transform has been used in action recognition [Gal+11]. As Gall's Hough Forest was used for action recogntion, and due to the importance of this algorithm in object detection, details will be provided in Section. 4

These tasks show that the HT is used for many real-life applications. As we planned to work on different computer vision tasks, the Generalized Hough Transform is a relevant algorithm to study. Hough Transform in tracking and detection context will be presented in chapters dedicated to these tasks, where we will position our works in Hough-based trackers and detectors.

2.3 Conclusion

This chapter was divided into two parts:

- First, we presented some visual features. We defined visual features as those that can be used to describe an image. We considered two families of visual features: color-based ones, and shape-based ones. In both cases, we defined low-level features (color histograms and scaled derivatives), to which we aim to limit our representation. Those features are fast to compute, and easy to parallelize. Moreover, they serve as a base for features with higher power of representation (color statistics, color attributes, superpixels, HOG, keypoints). We also presented some computer vision tasks that can be done with these features

- Second, we focused on one popular algorithm in computer vision: the Hough Transform. Originally presented in a patent to track particles in bubble chambers, it was formalized by Duda and Hart, then generalized by Ballard to detect any kind of shapes. After presenting some advantages and limits of the (Generalized) Hough Transform, we made a literature review to show how people tried to correct its weaknesses and how versatile it is, in terms of applications and adaptivity to different feature spaces

This chapter served to define the framework of the thesis: working only with color pixels and derivatives for the feature space, and using the Generalized Hough Transform as a spatial pooling mechanism.

In the two next chapters, we will focus on our works on object tracking and detection. The aim will be to see how far we can go with the constraints we set, and to analyze the benefits and limitations of our approach.

Chapter 3

Object Tracking

Contents

3.1 Definition 50
3.1.1 Tracking conditions 53
3.1.2 Difficulties 54
3.1.3 Conclusion 55
3.2 Literature review 56
3.2.1 State-of-the-art 58
3.2.2 Hough Transform for Object Tracking 64
3.3 Combining color histogram and Gradient for tracking 67
3.3.1 Backprojection map 68
3.3.2 Combining GHT and Particle Filter 69
3.3.3 Transitional tracker 73
3.3.4 Final tracker 77
3.4 Results 84
3.4.1 Implementation details 84
3.4.2 VOT datasets 86
3.5 Conclusion 111

The first studied computer vision task is object tracking. We chose this task for several reasons:

- Basically, object tracking consists in following the target object, in any kind of background. To do so, modelling the target can be done in different ways: by using features to describe it accurately, or using features to discriminate the target from the background

- The target's aspect can dramatically change over the sequence, and the context can complicate the task (camouflage or occlusion issues for example). The chosen feature space and the tracking algorithm have then to be flexible, to cope with these changes

- In some applications (Human Computer Interaction for example), the time consumption is a critical issue. In this condition, reactive trackers are important, and then, we consider that tracking is an interesting field of study, as it has to combine accuracy and speed

Those reasons make us believe that designed features should have a certain power of representation, be able to separate the object target from the background and be able to cope with object or context changes. Ideally, they should also be fast to compute and exploit.

The tracking task is one very old issue in computer vision, and has not been solved yet: academic datasets [KPL+]; [Kri+15]; [WLY13] are regularly proposed in order to test and evaluate algorithms.

Many applications rely on an accurate (and sometimes reactive) system of tracking:

- Surveillance: the aim can be to follow the trajectory of one potential threat. One example can be the task to track one person into a crowd of people. Occlusion, similar shapes (other humans) are some possible present difficulties in the scene

- Medical imaging: Tracking can be used to study the variation of shape of some organs. The problem is complicated, as, compared to sequences taken in urban scenes from a classic camera, images are usually noisy, and pixel resolution may not be sufficient for accurate tracking

- Augmented reality: the principle is to follow the position of some detected shapes or patterns, and then add some virtual objects in the scene. The problem is all the more difficult that given the point of view, the shape can be distorted

- Human-Computer Interaction: some applications can require gesture recognition, for which following diverse parts of the body over the time is an important task that should be done accurately. Moreover, real-time constraints are critical issues in this case

In this chapter, we will explore the field of object tracking, and detail our contribution in this area. The aim of our work will be to propose an accurate and robust tracker only based on gradient (used for the GHT) and color histogram (used for a simple foreground/background segmentation). By using these low-level features, we aim to propose a very fast tracker (above real-time), but still accurate and robust, given the constraints in terms of feature space and algorithms. We are working on the most generic conditions as possible: static or dynamic background and camera, one unique rigid or deformable target object (without abrupt motion change). We will then divide this chapter into four parts:

1. Object tracking is a very rich and complex problem. Then, in the first section, after defining the task, we will detail some properties to classify the task, and then, some traditional difficulties of object tracking. The aim will be to define the category of trackers we will work on

2. The second section will be a literature review. First, we will deal with different surveys, to understand their method for classifying trackers. Second, we will present some algorithms, from very popular trackers, to state-of-the-art ones, and to finish, a review of fast trackers. Third, we will end by detailing some modern Hough-based trackers

3. We will present our work, starting from our initial tracker [TM15], explaining its weaknesses, and how we solved them to propose our final tracker. The objective of our work was to propose a light, but effective tracker, built from very low-level features (gradient and color histogram exclusively). The final proposed tracker [TM17] will serve as a base for studies of the impact of used features

4. As our tracker has proven to be competitive on academic datasets [KPL+]; [Kri+15], we will dedicate a section to experiments and evaluation. The first part of the section will deal with implementation details (optimization, chosen parameters). Then, we will detail results obtained on modern datasets. We will also see the impacts of the limits we have set for our work (use of only low-level features) and study the impacts of different features

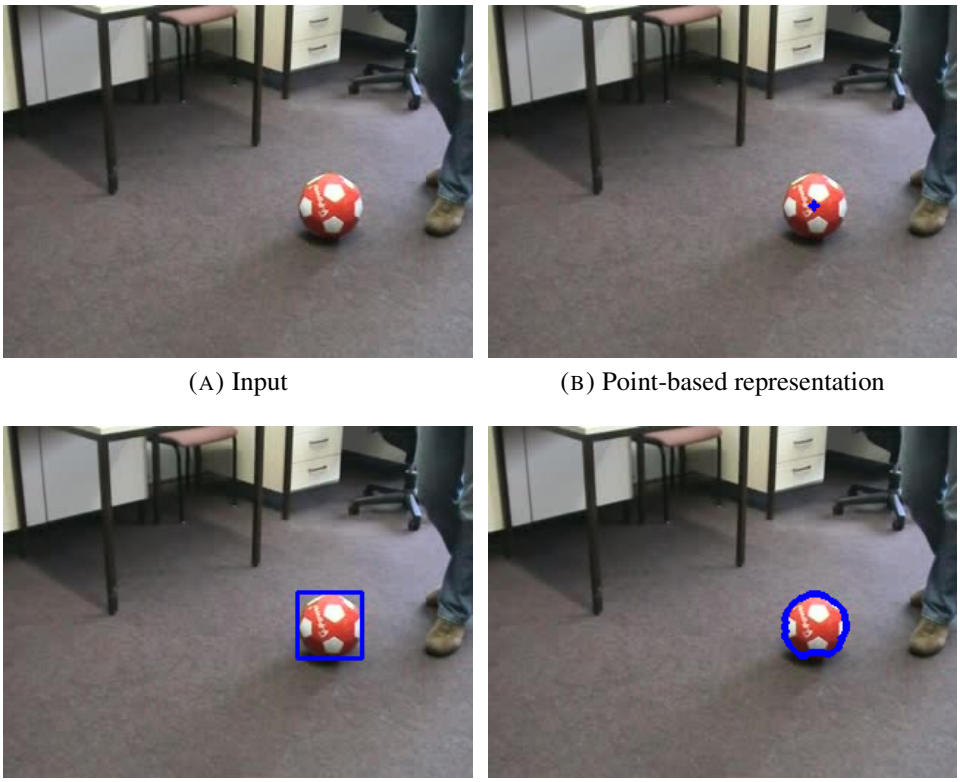
3.1 Definition

The first step is to define the problem. Basically, as suggested by [YJS06], object tracking consists in estimating the trajectory of a target object $\mathcal{O}$ during a sequence $(\mathbf{I}_t)_{t>0}$. In physics, the trajectory represents all positions taken by an object over the time. We think that this definition is limited, and we prefer using a more general definition proposed by Comaniciu in [CRM03], by considering that object tracking consists in estimating the *state* of an object (the target) over a sequence of images. This "state" may have several meanings, according to [YJS06]:

- The object's position. In this case, the object is represented by a reference point (such as its center). At this level of representation, the only possible movement that can be estimated are translations in x- and y- directions. Tracking a car (supposed not deformable) moving in a straight line with a camera placed orthogonally to the road is a suitable example of position only tracking. A pedestrian represented by its centroid, and filmed from a camera placed orthogonally to his movement, is also a case of position tracking

- Its position, size and orientation. In this case, the target is represented by a simple geometrical shape (slanted ellipse or rectangle), and the aim of tracking is then to estimate object's translation, orientation and scaling (movement that can be identified to translation along the z-axis). This method of representation is more suitable to track deformable objects, or objects whose size and orientation are changing over the time. Tracking the same car as before with a camera fixed anywhere is an example of application. This level of representation is one of the most used in recent academic datasets ([KPL+]; [Kri+15]; [Sme+14]), and we will adopt it

- Its contour, defining the target boundaries. Region inside the contour is then called silhouette. This representation is much more flexible than the previous one, and can distinguish movements from some elements of the target (arm movement of a pedestrian for example). It requires a pixel-level accuracy, and possibly a higher computation time (for example, due to the use of a segmentation algorithm). Tracking pedestrian is a possible application of tracking by silhouette



(A) Input (B) Point-based representation

(C) Bounding-box based representation (D) Silhouette representation (obtained using GrabCut [RKB04])

FIGURE 3.1: Different types of object representation

Fig. 3.1 illustrates these three different types of representation. Their common point is that the target is represented by one unique element (point, shape, or silhouette). However, there are other kinds of representation, modelling the object as the combination of different parts (for example, a human body can be described by the combination of its head, torso, arms and legs). We will refer to [YJS06] for a more detailed description.

This unique element used for target representation (or combination of object's parts) implies the notion of object, that distinguishes our problem from other kinds of tracking. Let us consider the case of optical flow, i.e. the task of estimating the apparent motion of pixels in a video from a dynamic scene. It can be considered as point tracking. However, one important missing part is the notion of object: in a dynamic scene, one given pixel is not moving independently from other pixels, it usually belongs to a set of pixels that globally have the same trajectory.

Generally, object tracking algorithm is composed of three steps. The first one is computed only at the beginning, offline, while the two others

form the tracking itself and are processed alternatively, until the end of the sequence:

- Initialization step. The aim is to build a model of the target to track. It can be done offline, by considering a training set, which can be generated either by different images of the target, or by considering the first frame of the sequence (in this case, it is composed of one unique image). In the second case, the target can be selected by different methods: manually (bounding box selection) or automatically (motion detection, background subtraction or object detection)

- Tracking step. Given an object model and an image $\mathbf{I}_t$, the goal is to estimate the state of the target object $\mathcal{O}$

- Updating step. This step is not essential to define object tracking, but is present in most modern trackers. The goal is to provide the tracker the ability to cope with different tracking issues (see Section. 3.1.2), by updating the model, adapting it to context changes over the sequence. This step is tricky as a tracking failure implies a wrong update, causing errors in next frames

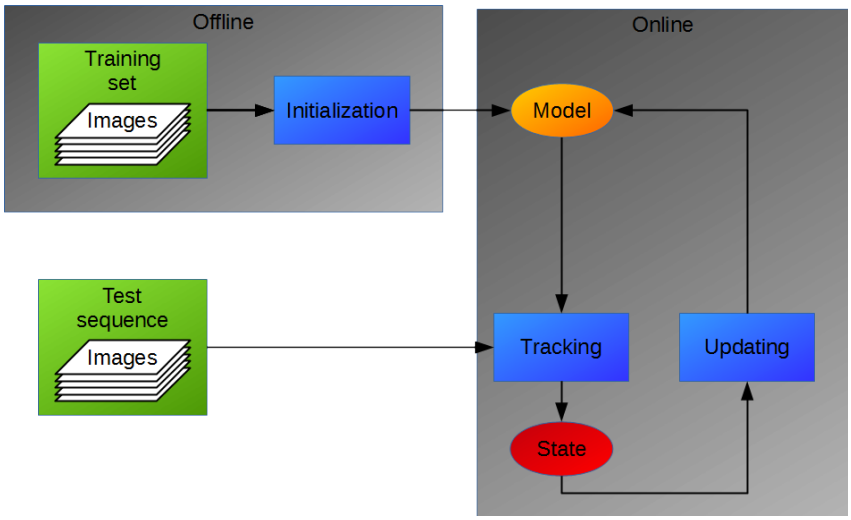


FIGURE 3.2: Diagram of a generic tracker.

Even with the definition of tracking proposed by Comaniciu [CRM03], and by the description of the chain of object tracking (illustrated Fig. 3.2), we can still subdivide the task into different categories. In the next part, we will deal with different hypotheses that can provide us a method to categorize the object tracking task.

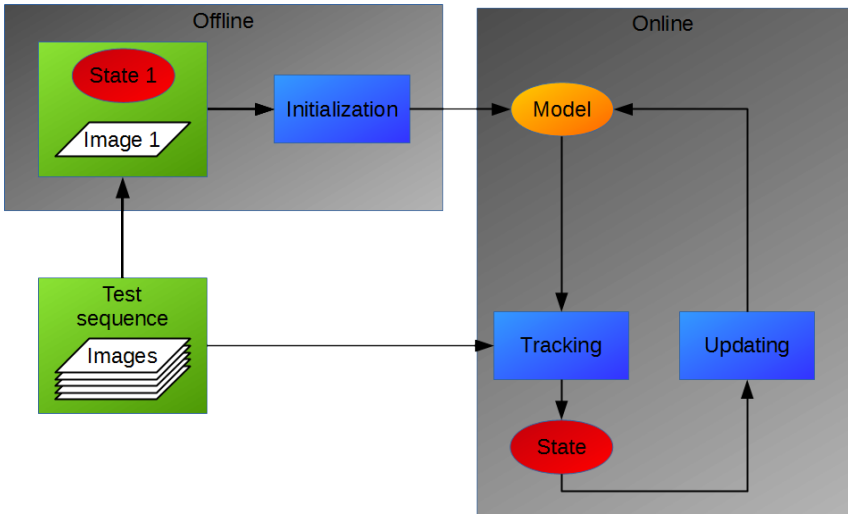


FIGURE 3.3: Diagram of model-free tracker.

3.1.1 Tracking conditions

Tracking is a very rich problem, and each tracker is usually designed for specific conditions. In this section, we present some conditions for which object tracker can be designed. The goal is not to be exhaustive, but rather to define the conditions in which we will work:

- Single or multi-camera. The problem of multiple camera tracking is more complex than the single camera one, and requires more complex algorithms: multi-view model, method to transfer the model from one camera to the other. However, it is potentially more robust to occlusion than the single camera tracking (better coverage of the scene)

- Single or multiple targets. Multiple targets can mean tracking different classes of objects (each one manually selected for instance). This case is an extension of single object tracking, but with different instances. We prefer defining multiple target tracking as the task of tracking different instances of one class of object (pedestrian, or car) in a given sequence. This kind of problem requires an offline learning of this class, and is close to the object detection task. In both cases, we have to consider new challenges, like target crossing

- Genericity. Ability to track any kind of object. Opposed to object-specific tracker (such as pedestrian trackers), for which object tracking is very close to object detection, because, in this case, the use of training set can improve tracker's performance

- Model-free. This hypothesis means that the model initialization is done only with the first frame of the sequence, for which the target state is given. Fig. 3.3 illustrates the execution of model-free trackers

- Short-term *vs* long-term tracking: following Kalal's definition [KMM12], long-term tracking tackles the problem of following an object on a long video, and has to indicate the presence or not of the object on the filmed scene. The problem is complicated, as, when the target reappears, its appearance can change. Usually, a long-term tracking combines tracking and detection routines

- Speed of the object: recently Rozumnyi [Roz+16] formalized the notion of fast moving object in tracking case by considering that, an object is moving fast when, in the exposure time interval, this object is moving at a distance greater than its size. Difficulties are then related to visual target shape (blurred aspect) or impossibility to use prediction models designed for "slow" targets

This list of hypotheses is non-exhaustive. However, these different properties set the context in which we will propose our tracker in Section. 3.3. Indeed, trackers we will present are designed to track single arbitrary and "slow" target, with a single camera. They will also be model-free, and designed for short-term tracking.

3.1.2 Difficulties

In this part, we will focus on difficulties and constraints inherent to object tracking, to understand why the problem is not solved yet. These difficulties are independent of the application context (especially those linked to the computation time), and to tracker properties mentioned in Section. 3.1.1. They can cause drift in tracking. Among these constraints, we can cite:

- Object geometrical transformation, such as scaling (translation over the z-axis) or rotation (in the image plane or not). For bounding-box object representation, these issues imply that the bounding box orientation and size may change (with possible aspect-ratio changes)

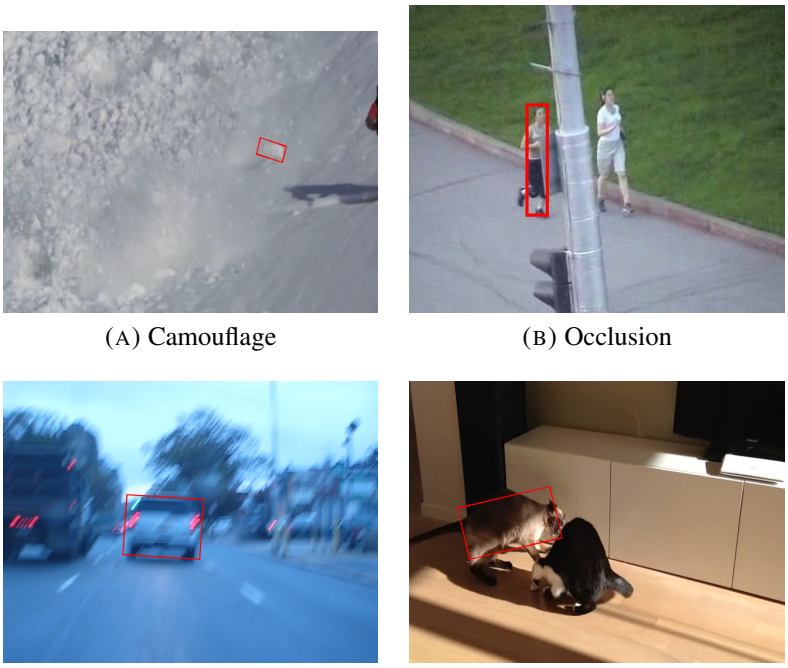
- Object deformation, or aspect change. It is notably true for non-rigid objects (pedestrian for example). In our case (bounding-box representation), aspect-ratio may not be constant
- Illumination change, which can create reflections in objects, appearance (or disappearance) of cast shadows. Shape-based trackers are robust to illumination change (up to a certain level), unlike color-based trackers
- Partial or complete occlusion. A partial occlusion can be caused by the background, or by the object itself (consider for example a walking human). In this case, modeling the target with several elements is a better choice than with one unique. For total occlusion case, the problem can be decomposed into two sub-problems: the loss of the target and its recovery. This case is then close to long-term tracking problem
- Motion change of the camera. While for static camera, we can provide a background subtraction function to reinforce the tracker, for moving camera, the problem is more difficult, as every element of the scene is moving
- Complexity of the background. Moving objects in the scene, object visually similar to the target (camouflage), presence of several objects are among the problems caused by a complex background
- Length of the video. The longer the video is, the higher the risk of drift and failure is. In studied benchmarks, most sequences are very short (less than 1000 frames)

Fig. 3.4 illustrates some common difficulties in tracking context.

3.1.3 Conclusion

In this section, we aimed to introduce object tracking and set the context of our work. We saw that even though the definition is clear, different kinds of tracking context exist, depending on the method of representation, the different possible properties, and the context they are applied to.

The next section will be an overview of the literature. At this point, as our work on tracking will be to track bounding box represented target over the time, for the rest of this chapter, let us define notations that we will use



(A) Camouflage (B) Occlusion

(C) Camera motion (D) Illumination change

FIGURE 3.4: Some difficult frames from VOT2014 and VOT2015 datasets

for tracking context. Given a sequence $(\mathbf{I}_t)_{t\geq 0}$, the aim of the tracker will be to estimate, at the frame t, the object bounding box $B_t = \{c_t, w_t, h_t, \theta_t\}$ such that:

- c_t is the bounding box center
- w_t its width
- h_t its height
- θ_t its orientation

For any image $\mathbf{I}$ and any set of pixels B , let $\mathbf{I}_{|B}$ be the sub-image of $\mathbf{I}$ restricted to B. For commodity, we will also denote $\{c_t, w_t, h_t\} = \{c_t, w_t, h_t, 0\}$.

3.2 Literature review

In this section, we focus on object tracking as an algorithm, by starting with a literature review, presenting some state-of-the-art algorithms (in terms of accuracy and speed). The aim is not to propose a full coverage

of the tracking problem, but rather to propose a summary of how trackers can be classified, to understand in which category our work resides.

As mentioned before, object tracking is a very popular task in computer vision (according to the VOT committee [Kri+15], in major conferences in computer vision (ICCV, ECCV, CVPR...), each year, about 40 papers deal with object tracking), and many algorithms exist in the literature. Yilmaz [YJS06] proposed a method to classify object trackers, inspired by his classification of object representation:

- Point-based trackers, in which objects to track are represented by a sparse set of points. At each frame, the motion of these points are evaluated (optical flow estimation). This class of tracker is suitable for objects with complex shapes or largely textured object(with many keypoints to detect)

- Kernel-based trackers, deriving directly from the second level of representation defined in Section. 3.1 (geometrical shape). The tracking problem is then similar to estimating of the motion of the target (translation essentially, but it can also include scale and orientation estimation) represented by a shape (bounding box, ellipse). One popular type of kernel-based tracker is based on the cross-correlation between a template modeling the object target and some areas of a tested image. Our trackers belong to this category

- Silhouette-based trackers: deriving from the third class of representation defined previously. It suits complex objects (star shaped objects for example), or when it is necessary to have an accurate level of description. Silhouette tracking can be used for applications such as human gesture recognition, requiring to track different parts of humans body, and send results to a classifier. Yilmaz distinguishes two classes of silhouette-based trackers: one based on the object' shape, the second based on the contour tracking

All these categories also include subcategories (Yilmaz for example considers two classes of Kernel-based trackers, one based on template, another based on multiview models). One interest of this classification is the domain of use of each class of tracker: according to the application, developers should prefer one type of tracker to the others.

This classification served as a reference for Cannons' survey [Can08]. However, one difference between the two surveys is the first class of

tracker: Cannons extends point-based trackers to discrete-feature based trackers (including for example trackers based on group of edges).

Yang [Yan+11] proposed a review of object tracking by considering an algorithmic point of view. Indeed, he supposed that a tracker needs a feature descriptor step to effectively track a target, and choosing the right feature for the right context is necessary. Then, this chosen feature is used to train an online learning algorithm, used to cope with object's (and sometimes context's) appearance change (updating the model), and for the decision (state estimation).

In the two next sections, we will propose an overview of different different trackers from the literature. Then, we will present some Hough-based trackers.

3.2.1 State-of-the-art

All trackers presented in this section belong to the same category as ours: model-free trackers without recovery function, and aiming to track object represented by geometrical shape. We will elaborate on Hough-based trackers in the next part.

The first tracker we will mention, due to its relation with our work [TM15], is the Particle Filter, proposed by Isard and Blake [IB98] in tracking context. It is a method coming from statistics, and used to estimate some parameters of a dynamical system, by using a set of observations. Isard and Blake's CONDENSATION algorithm [IB98] is composed of three steps:

1. Observation: each particle, defining a hypothetical parameter set, is associated to a weight, related to a confidence value (measured by an observation). Then, given the set of particles and their measure of confidence, target's state is estimated by an average operator

2. Resampling: To ensure a certain quality of particle set (for example few particles with low weight and a set of particles covering a certain area in the parameter space), a resampling step can be useful, by discarding some particles and adding others

3. Propagation: given a dynamical model, all particles are spread within the parameter space. A simple propagation model can be applied by this equation:

$$X_t = \mathbf{A}_t \cdot X_{t-1} + B_t \tag{3.1}$$

with X_{t-1} a vector representing one particle at $t-1$ (of size $n_s \times 1$, n_s being the dimension of the state space), X_t the same particle at t, $\mathbf{A}_t$ a matrix of size $n_s \times n_s$ modelling a dynamical model, and B_t a $n_s \times 1$ vector modelling a noise

Particle filter is flexible in terms of feature spaces: Isard and Blake modeled object target using edge features, by modeling them with Bézier curves [Béz66], while Nummiaro [NKMVG03] modeled it using color histogram (confidence measure is related to Bhattacharyya coefficient Eq. 2.7). Pérez [Pér+02] also exploits color-based Particle Filter for tracking context, by exploiting a richer model of representation: the target is no longer represented by one image, but by a set of sub images, obtained by partitioning. We can also cite [Bre+09] who realize a multi-pedestrian tracker based on particle filter and a detector. Its dynamical model during the propagation step is computed according to pedestrian position and motion. [Dub15] proposed a coverage of tracking problem, addressed with Particle Filter.

The second tracker we mention in this thesis is the Mean shift tracker. Originally, Fukunaga [FH75] designed it for maxima search in density function. Comaniciu popularized it in image processing by showing its versatility and applying it for image filtering and segmentation [CM02]. However, we will put our interest in his adaptation for tracking context [CRM03]. Indeed, this algorithm has proven to combine accuracy and speed (Comaniciu mentioned that it can run at 150 fps in its optimized version, on a 1.0 GHz computer), and is still inspiring some modern trackers (especially [VNM13] that is one of the fastest algorithms in [Kri+15]). The algorithm is based on two concepts: the *target model* and *localization*. In its original form, the Mean shift does not include updating process. First, the target model is built at the frame 0, and using the first object center c_0. The object is modelled by an ellipsoidal shape S_0. Comaniciu also considered one feature space: even though his algorithm is presented using color space, it is usable with other feature space (such as shape or texture). At the first frame, the target is modelled using a weighted histogram. To do so, he defines a kernel function k, and a bandwidth h. These two parameters are chosen to set high weights to pixels close to object's center (more prone to belong to the target), and used to build object

histogram $H_{\mathcal{O}} = H_{\mathbf{I}|_{S_0}}$:

$$H_{\mathcal{O}}(b) = C \cdot \sum_p k(\frac{||p-c||}{h}^2)\delta(\mathbf{I}(p) - b) \tag{3.2}$$

with C a normalizing term (to set the sum of the histogram to 1.0). Eq. 2.6 is a particular case of this equation, with k a function equal to 1.0 in $\mathbf{I}|_{B_0}$ and 0 outside. Then, at the frame t, the goal is to estimate the shape S_t with the same dimension and orientation as S_0. The problem is then to estimate object center, and Comaniciu considers that this center c_t is obtained by localizing the shape S_x centred in x for which the Bhattacharyya coefficient B of $H_{\mathcal{O}}$ with $H_{\mathbf{I}_t|_{S_x}}$ is optimal:

$$c_t = \underset{x}{\operatorname{argmax}}(B(H_{\mathcal{O}}, H_{\mathbf{I}_t|_{S_x}})) \tag{3.3}$$

While it is possible to solve Eq. 3.3 with brute force search, Comaniciu makes the supposition that c_t is close to c_{t-1} ("slow" tracker hypothesis). The Taylor expansion can then be computed for Bhattacharyya coefficient Eq. 2.7 for potential centers close to c_{t-1}. From this Taylor expansion, and considering [CM02], the Mean shift vector is defined by:

$$ms(x) = \frac{\sum_p p \cdot \omega_p \cdot k'(\frac{||x-p||^2}{h})}{\sum_p \cdot \omega_p \cdot k'(\frac{||x-p||^2}{h})} \tag{3.4}$$

where:

$$\omega_p = \sqrt{\frac{H_{\mathcal{O}}(p)}{H_{I_t|_{B_x}}(p)}} \tag{3.5}$$

At a frame t, given $x_0 = c_{t-1}$, the algorithm calculates the series (x_n) such as $x_{n+1} = ms(x_n)$. (x_n) is supposed to converge to c_t (Even though the convergence in the general case, with any kernel function k, has not been proved yet [Gha15]). Intuitively, c_t serves as an attracting point, and Eq. 3.4 will shift the center candidate to high weight areas. Comaniciu also presents methods to optimize the algorithm (in particular case, Eq. 3.4 is not computed directly, but with a certain approximation) and to improve the accuracy or scale adaptation. Mean shift tracker has many advantages: its simple implementation, its versatility (in terms of application, and in terms of ability to take as an input different feature spaces), and its speed. Many trackers exploit these assets, to propose extended version of the Mean shift. We mention Bradski's CAMSHIFT [Bra98] designed for face

tracking, who supposed that skin color projected into the Hue space is invariant, and then used this color channel for his tracker. He also estimates face pose (size and orientation estimation) by using statistical moments. We can also mention Birchfield [BR05] who applied Mean shift using spatiograms (see Section 2.1.1 for details about spatiograms) by using a more complex metric than Bhattacharyya coefficient, based not only on color histogram, but also on color centroids and standard deviations). If we focus on state-of-the-art trackers, Vojir's extension leads to high speed tracker (more than 100 fps) with decent accuracy and robustness. Vojir's enhances the mean shift in two ways:

- For scale estimation, he uses a formulation aiming to maximize the ratio between the Bhattacharyya coefficient of the model with the target candidate and the Bhattacharyya coefficient of the model histogram with the background
- He also improves scale estimation by computing a backward tracking (tracking from t to $t-1$) and study the consistency of the estimated scale

Regarding to state-of-the-art trackers, we can use Visual Object Tracking annual challenges [Kri+13]; [KPL+]; [Kri+15] as a reference. Details linked to evaluation criteria will be provided in Section. 3.4. However, at this point, the ranked participants of these challenges provide current tendencies in object tracking.

For the first VOT challenge [Kri+13], the winner proposed a tracker derivating from STRUCK [HST11], which belongs to the family of discriminative trackers (the winning tracker paper is not available). To do so, Hare uses a SVM [CV95], which is trained online for tracking context. All its (positive and negative) samples are represented into a Haar feature space.

Danelljan's DSST [Dan+14a] was ranked first in the second edition of the VOT challenge [KPL+]. It belongs to the class of correlation-based trackers, providing accuracy and decent speed. It was inspired by Bolme's tracker [Bol+10], MOSSE. The principle of correlation-based trackers is to train a filter f, by first considering an image $\mathbf{I}_0$, a ground truth GT_0 and generating a set $(I_{0,i})_{1\geq 1}$ of sub-images from $\mathbf{I}_0$, associating each image to an output image $\mathbf{O}_i$ (Gaussian peak localized at target center). These images $I_{0,i}$ are defined by a bounding box $GT_{0,i}$, such that $GT_{0,i}$ is obtained

by randomly perturbing parameters of GT_0, and $I_{0,i} = I_0|_{GT_{0,i}}$. Then, the correlation filter f is trained by optimizing:

$$min_f(\sum_i ||\mathbf{I}_{0,i} * f - \mathbf{O}_i||) \tag{3.6}$$

This equation, computationally costly, can be accelerated in the Fourier space, as the convolution operation in image space is equivalent to a point-wise product in Fourier space. Then, Eq. 3.6 leads to:

$$min_{\hat{f}}(\sum_i ||\hat{\mathbf{I}_{0,i}} \cdot \hat{f}^* - \hat{\mathbf{O}}_i||) \tag{3.7}$$

where $\hat{\bullet}$ is the Discrete Fourier Transform operator, and $\bullet^*$ the transpose-conjugate operator. Bolme also considered filter updating. Danelljan's extension [Dan+14a] is a more accurate tracker using ideas from his previous work [Dan+14b] in which he designed correlation filters for multiple features. In [Dan+14a] case, PCA-HOG [Fel+10] is used. Another improvement is on the scale estimation, for which Danelljan designs 3D filters (the third dimension being the scale). VOT2016 winner [Dan+16] belongs to this family of tracker, and its formulation provides to the tracker a way to integrate feature maps at different resolutions. STAPLE [Ber+16] is a tracker with a structure close to ours (combining complementary and independent color-based and shape features trackers) and running above real-time speed. Its shape tracker is based on correlation filter, using HOG features [DT05], while its color-based tracker is based on color histogram. Due to their similarities, we will provide a detailed comparison of this method and our work in Section. 3.4.1.

Recently, Deep Learning trackers were proposed, and were well ranked in the VOT challenge [NH16]; [NBH16]. One early tracker using Convolutional Neural Networks was proposed by Fan [Fan+10], for human tracking (an offline process consisted in training the networks using video containing human heads, to make them learn some spatial and temporal features). DeepTrack [LLP16] is a generic tracker using CNN, without offline training. MDNet [NH16] is a CNN-based tracker which won the challenge VOT2015 [Kri+15]. Its offline training consists in learning networks divided into different domains. Then, some information independent of the domains is extracted, to build features for the network. Then, during the tracking step, a bounding box regression technique [Fel+10] is computed to estimate object state, followed by an updating process.

The main weakness of CNNs-based trackers is the high computation time (those previously cited run at less than 5 fps on a GPU). However, recently, Held [HTS16] proposed GOTURN, a CNN-based tracker able to track at 100 fps. To do so, the network is trained offline in order to detect motion and appearance change. At runtime, the absence of online training, combined with a network running feed-forwardly allows the tracker to run at 165 fps with a high-end GPU, but only at 3 fps on a CPU. Moreover, for objects absent in the training set, the tracker suffers from a little loss of performance.

In any case, tendencies in object tracking are to propose trackers with high time consumption operations (correlation filters, classifiers, CNNs) and complex features (HOG). However, we still find some accurate and light trackers, such as Possegger's [PMB15], using only color histogram. Some details will be provided in Section 3.3, but one key element of this algorithm is to reduce influences of distractors around object's position to avoid drifting.

Now, in terms of fast trackers, we already mentioned Vojir's adaptation of Mean shift [VNM13] as one of the fastest trackers from all the VOT challenges. He also uses color histogram as a model. Matas [MV11] proposed a real-time tracker, based on a point tracker called MedianFlow [KMM10]. The principle is to model the target at the frame t by a set of points (x_t^i) and track them using optical flow estimation [Shi+94]. Each point x_t^i that moves to a point x_{t+1}^i at frame $t+1$ is supposed reliable if this point x_{t+1}^i moves to a point close to x_t^i from frame $t+1$ to frame t (backward tracking). By discarding 50% of the unreliable points (x_{t+1}^i) and computing the bounding box of the remaining set, Kalal [KMM10] managed to propose a high-speed tracker. Matas [MV11] improvement concerns two steps: the point sampling was originally done using a regular grid, while Matas proposed to subdivide images into different regular cells, and then draw one point per subdivision. The second improvement is for the point tracking failure estimation: Matas adds a measure of consistency, and a Markovian model. Another state-of-the-art real-time tracker was proposed by Henriques [Hen+15], who made an extension of his work [Hen+12]. The method relies on a discriminative classifier, which is trained by sampled patches. The novelty of his work is the fact that these patches are sampled using translation and scaling. In that way, due to the redundancy of information, the trained classifier can be formally written with circulant matrix that have nice properties in Fourier domain,

which allows the classification to be done quickly.

In this section, we made an overview of different methods of tracking. After presenting an overview of Particle Filter based trackers, we presented the Mean shift [CRM03], a tracker serving as a base for a very efficient tracker [VNM13]. In terms of state-of-the-art trackers, current methods are based on correlation filters. More recently, CNNs-based trackers were developed, and achieved high accuracy. We then finished by citing some real-time trackers. In the next part of this chapter, due to the importance of the GHT on our work, we will focus on Hough-based trackers.

3.2.2 Hough Transform for Object Tracking

Even though Hough Transform [Hou62] and its generalization [Bal81] have been designed to detect shapes, several authors adapted them for object tracking.

Intuitively, directly applying the Generalized Hough Transform (GHT) presents a certain interest. Indeed, in case of translations parallel to the image plane, a maxima search in the Hough Space is usually sufficient to estimate the target location: all pixels are translating identically, and so the target peak is moving at the same speed and direction. Furthermore, considering that the original GHT only requires gradient computation, it is robust to illumination change.

However, in tracking context, pure translation in the image plane is very rare in real cases. Moreover, when object motion is too fast, target shape is blurred, resulting to a wrong position estimation of the GHT (loss of structure). If we follow Ballard's suggestion, by voting in n_s different scales and n_o different orientations, then the memory consumption is multiplied by $n_s \cdot n_o$. The lack of method to update target's model (the R-Table) is also a flaw for the original GHT in tracking context (as it was designed for shape detection, and not for tracking).

For these reasons, the simple Generalized Hough Transform is not usable as is for object tracking. However, the principle of considering a set of elements from an image (pixels, patches...), and making them vote for a set of parameters, definitely makes sense for tracking context.

Sato [SA04], for instance, combined Hough Transform and temporal windowing to track pedestrian, filmed in lateral-view. He starts by binarizing all images of a given sequence $(\mathbf{I}_t)$ with a background subtraction method. Then, he extracts some *standing objects* corresponding to regions

intersecting an area around the horizon line (depending on the height of the camera), and with a height below a threshold. These standing objects correspond to potential pedestrians. This hypothesis is restrictive, as it is designed to detect pedestrian walking at the same altitude, and those who walk on another plane can not be tracked. Finally, the Hough Transform is applied by considering that a pixel from these standing objects follows a certain trajectory (supposed to be a straight line), at a certain speed (supposed constant). A blob representing a pedestrian is then extracted by considering that all pixels from the same pedestrian follow the same trajectory at a constant speed. Sato also extended his tracker to recognition of simple interactions between humans, such as one person following another one, one person stopping in front of a second one, etc. However, this tracker is very limited: experimental conditions are very restrictive and can not be adapted to a more general context (camera motion, pedestrian walking at different planes).

Recently, Hua in [HAS15] proposed a tracker also adapting Hough Transform. His tracker has proven accurate on VOT2015 dataset [Kri+15]. First, Hua detects potential candidates (represented by bounding-boxes) using a HOG-based [DT05] detector. Second, he applies a Hough Transform to estimate the geometrical transformation of the target. After estimating the optical flow, each pair of points votes for a geometrical transformation, giving a first score of detection confidence. Third, two scores, one based on object edges and the second on motion, are generated, and the three computed scores are used to estimate the new state (bounding box). However, this tracker is very slow, far from the real-time criterion (less than 7 fps). Moreover, the Hough Transform is only used in order to validate or reject potential states.

If we deal with the Generalized Hough Transform, several GHT-based trackers have also been proposed. Pixeltrack [DG13] is a very-fast tracker (more than 100 fps) proposed recently, very close to the GHT. Its high-speed is mainly due to very low-level operations, and to a fully optimized code.[1] At each frame t, the first step consists in computing a GHT to estimate a potential object center x_h. Unlike Ballard's method [Bal81], Duffner indexes his R-Table not only with gradient orientation, but also with HSV color values. Then, he proceeds to an operation called *back-projection*, that we will detail in Section 3.3. Duffner proceeds to a fast

[1] http://u0016403263.user.hosting-agency.de/research_pixeltrack.html

segmentation using color models of the background and the foreground, and computes the centroid of this map x_s to get a second potential center. Finally, the target center is estimated as the weighted average of x_h and x_s. The weight of these two candidates depends on a confidence measure of the Hough Transform. Finally, the update process is done by considering a segmentation map, computed from the color segmentation and the backprojection map. This algorithm served as a reference for our work, as it presents a certain trade-off between accuracy and speed.

Maresca [MP13] also proposed a tracker using the GHT, with a higher level of representation than gradient features, based on keypoints features (notably SIFT [Low99] or ORB [Rub+11]). The R-Table is replaced by a codebook (built using positive and negative samples), and correspondences are found using a k-nearest neighbor (K-NN) algorithm. At a given frame t, all detected keypoints are put into a K-NN algorithm to search for correspondences (this training is done during the tracker initialization and the updating process). Then, after some filtering operations (in order to discard some bad keypoints, such as those close to a negative samples) in the feature space, the GHT is realized by making detected keypoints vote for a position. Finally, a scale estimation is computed. MATRIOSKA is a decently accurate and robust tracker (it was among the first half of the competing trackers in VOT2014 [KPL+]), and using a keypoint-based representation has certain advantages (robust to illumination change, point of view change...); its speed is far lower than that of PixelTrack (about 15 fps), due to higher level-operations (detection and description of keypoints, k-NN). However, by combining MATRIOSKA with BDF [MP14], which is a variant of the Flock of Trackers [VM14], Maresca managed to propose a high-speed tracker.

Godec [GRB13] proposed a tracker based on the Hough Forest framework proposed by Gall [Gal+11] (a more detailed description of the Hough Forest is available in Section. 4). Like Maresca, the R-Table is based on a machine learning classifier: the Random Ferns. Godec however uses a patch-based representation (each patch is described using Lab color space, first and second derivatives and HOG features [DT05]) to train his decision forest. At each frame $\mathbf{I}_t$, each patch goes through all decision trees (the ferns), and then votes for different positions, according to the leaves. Then, to estimate the size and updating the model, Godec first proceeds to a backprojection operation, and then to a GrabCut segmentation [RKB04].

In terms of performance, the Hough Fern tracker is among the least accurate ones in [KPL+], and due to heavy computation operation (HOG evaluation and GrabCut segmentation), is also very slow.

In this part, we highlighted the limitations of the (Generalized) Hough Transform for object tracking. However, as many researchers proposed different trackers to compensate these weaknesses, we made a short review of recent Hough-based trackers. These proposed trackers can be very accurate and robust [HAS15] or very fast [DG13], and enhanced the Hough Transform at different levels:

- Some researchers applied the (Generalized) Hough Transform using a higher level of representation than the original method: Godec [GRB13] used HOG features, [MP13] used keypoints, [SA04] and [HAS15] used temporal features. Those works show that the Hough Transform is able to support high-level features, and even temporal ones

- Others used Machine Learning method to improve the voting process [MP13]; [GRB13]. Both papers changed the R-Table to a more complex data structure: a codebook indexed by keypoint descriptors for [MP13], random ferns for [GRB13]

- Others proposed to combine the Generalized Hough Transform with another model [DG13]

It turns out that the Hough Transform is a framework that can be improved at different levels for tracking purpose. In the next sections, we will present our methods, together with experiments realized on academic datasets.

3.3 Combining color histogram and Gradient for tracking

In this section, we will present our work in object tracking. This section will be mainly inspired by [TM15] and [TM17]. However, we plan not to make a raw copy of these two papers, and prefer studying these trackers and see their advantages and weaknesses.

This section will be divided into four parts:

1. First, we will present the backprojection operation, common to all approaches
2. Second, we will present [TM15], which combines the GHT and an adaptation of the Particle Filter
3. Third, we will present the weaknesses of the previous tracker, and explain how we partially solved them
4. Fourth, we will present [TM17], detailing how simple routines can lead to an effective tracker (see Section. 3.4)

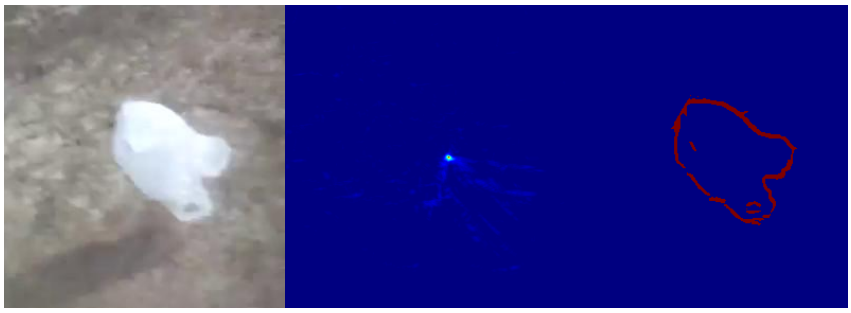
The common point of these three trackers is the exclusive use of gradient features to compute the Generalized Hough Transform, combined with color histogram. The problem will be then to combine these two bases to propose a decent tracker. In terms of performances, results and evaluation on academic datasets and computation time will be detailed in Section. 3.4.

3.3.1 Backprojection map

Backprojection map is one common point to the two trackers we proposed. In object detection, Razavi [RGVG10] uses this operation in many ways, and we will detail them more precisely in Section. 4. In this section, we will only present its use in tracking context. This operation, as used by Godec [GRB13] and Duffner [DG13], is used to determine the support of pixels which have voted for best object location. Formally, given a R-Table R, the Hough Transform $\mathbf{HT}$ from an image $\mathbf{I}$, the backprojection associated to the location x, denoted $\mathbf{BP}_x$, is a map null for every pixel p, except those which have voted for x:

$$\mathbf{BP}_x(p) = \begin{cases} \mathbf{HT}(x) & \text{if } \exists \vec{u} \in R(\theta_p), p + \vec{u} = x \\ 0 & \text{otherwise} \end{cases} \tag{3.8}$$

Duffner and Godec used this binary backprojection as a confidence map by selecting x such as $x = \max_p(\mathbf{HT}_p)$ (see Fig. 3.5). The formulation Eq. 3.8 differs from those we will use in our works. However, both formulations are based on the same idea, that if one pixel has voted for the peak, it is more prone to belong to the tracked target.



(A) Input (B) Hough Transform (C) Backprojection

FIGURE 3.5: Backprojection from *bag* sequence.

3.3.2 Combining GHT and Particle Filter

We present [TM15] in this part. This tracker combines the GHT, and an adaptation of the Particle Filter [IB98]. We aimed to use these two algorithms in their simplest form.

The first step is the model initialization. As our tracker belongs to model-free ones, it requires only the first frame $\mathbf{I}_0$ and the first bounding box B_0. The R-Table R is initialized using all pixels inside B_0 whose gradient magnitude is above a threshold ϵ_M. It is used as a shape model for tracking. Unlike the original GHT, the R-Table does not store displacements exclusively, but couples $(\vec{u}, \omega_{\vec{u}})$, where $\omega_{\vec{u}}$ is a weight associated to a displacement $\vec{u}$ (defined as in the original GHT). On the other hand, for the particle filter, let us define $H_{t=0}$ the initial normalized color histogram of $\mathbf{I}_0|_{B_0}$ as a color model. Let $N_p = 500$ be the fixed number of particles used in the whole sequence. At each frame t, one particle is defined by its hypothetical state $B_t^i = \{c_t^i, w_t^i, h_t^i\}$ (orientation is set to 0.0), and its weight ω_t^i.

Then, for tracking step, the GHT and the particle Filter are computed independently. On the one hand, as mentioned in Section. 3.2, the Particle Filter is divided into three steps, and in our case, computed that way:

- Observation: at each frame t, each particle i represents a potential state. Its weight is updated by computing the Bhattacharyya coefficient B between H_t (normalized color histogram model at frame t) and H_t^i, color histogram of $\mathbf{I}_t|_{B_t^i}$. Then, the particle weight is equal to $\omega_t^i = \exp\left(-\lambda \cdot (1.0 - B(H_t, H_t^i))^2\right)$, where $\lambda = 50.0$. This weight measure is inspired by Pérez [Pér+02] adaptation of Particle Filter in tracking context

- Resampling: the aim of the resampling step is to preserve a trade-off between coverage of the state space and proportion of particles with high weights. Arulampalam proposed different methods to resample the particle set [Aru+02]. However, as we aim to propose a simple algorithm, the particle set is at each frame fully regenerated using a multivariate Gaussian Process. For each particle, c_t^i is sampled from a 2D Gaussian random process centered in c_{t-1} (previous estimated state), and with variances $(c \cdot w_{t-1}^i, c \cdot h_{t-1}^i)$, with $c = \frac{1}{2 \cdot 2 \cdot \sqrt{2 \cdot \ln 2}}$. The denominator $2 \cdot 2 \cdot \sqrt{2 \cdot \ln 2}$ is linked to the full width at half maximum (FWHM) coefficient of a Gaussian of standard deviation σ: $FWHM = 2 \cdot 2 \cdot \sqrt{2 \cdot \ln 2} \cdot \sigma$. We choose these variance value to ensure that most particle centers are close to c_{t-1}. For the scale parameters, (w_t^i, h_t^i) are drawn using a 2D Gaussian centered in (w_{t-1}^i, h_{t-1}^i) and of variances $(\beta \cdot w_{t-1}^i, \beta \cdot h_{t-1}^i)$, with $\beta = 0.05$.

- Propagation: as the resampling process regenerates all particles, there is no propagation step in our tracker. So, given Eq. 3.1, at each frame, we simply have $\mathbf{A}_t = Id_4$ (the identity matrix of size 4) at each frame, and B_t the null vector

Let us then define $\mathbf{OP}$ the observation map, such that:

$$\mathbf{OP}(x) = \begin{cases} \omega_t^i & \text{if } \exists i \text{ , } c_t^i = x \\ 0 & \text{otherwise} \end{cases} \tag{3.9}$$

As the particle filter is based on estimating the density of probability at different states (i.e. different particles), $\mathbf{OP}$ can be seen as a sparse representation of the probability to find the target, at different states.

On the other hand, the GHT is computed normally, with n_g orientations to index the R-Table R, except that, contrary to Eq. 2.28, each pixel votes according to each displacements $\vec{u}$, with a weight $\omega_{\vec{u}}$:

$$\mathbf{HT}(p) = \sum_q \sum_{(\vec{u}, \omega_{\vec{u}}) \in R(\theta_q)} \omega_{\vec{u}} \cdot \delta(p, q + \vec{u}) \tag{3.10}$$

where θ_p is the quantized gradient orientation of p. From the GHT, we compute a backprojection map different from Eq. 3.8:

$$\mathbf{BP}(p) = \max_{\vec{u} \in R(\theta_p)} \mathbf{HT}(p + \vec{u}) \tag{3.11}$$

This backprojection is softer than Duffner's and Godec's formulations Eq. 3.8. Indeed, they backproject the peak, while in our case, all voting pixels are strictly positive in $\mathbf{BP}$. However, in both cases, the aim of the backprojection map is the same: the higher the value of one pixel of the backprojection, the more probably it belongs to the target. Fig. 3.6 illustrates this soft backprojection.

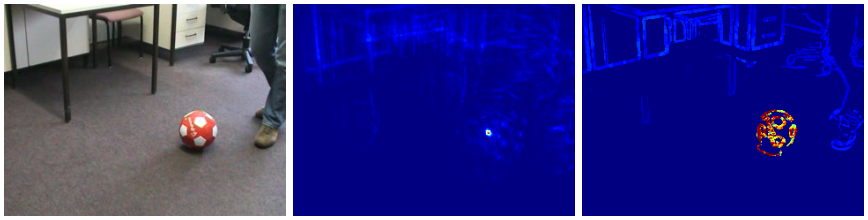


FIGURE 3.6: Backprojection obtained by Eq.3.11.

Then, for state estimation, we need to combine the two outputs $\mathbf{BP}$ and $\mathbf{OP}$. One problem is that $\mathbf{OP}$ is a sparse representation of a probability map, and then needs to be made denser. Fitting it with a probability function, such as a Generalized Gaussian, is possible, but, for efficiency reasons, we preferred some lighter (but less accurate) methods. Indeed, we choose to make the map denser by using a morphological dilatation, with an elliptic structuring element $\mathbf{SE}$ of size $(\frac{w_{t-1}}{{N_p}^{\frac{1}{3}}}, \frac{h_{t-1}}{{N_p}^{\frac{1}{3}}})$. Let us denote $\delta_{\mathbf{SE}}(\mathbf{OP})$ the obtained map.

Finally, let us define the fusion map $\mathbf{B}$, such that:

$$\mathbf{B} = \mathbf{BP} \cdot \delta_{\mathbf{SE}}(\mathbf{OP}) \tag{3.12}$$

$\mathbf{B}$ is then normalized to $[0, 1]$, and thresholded.

This map serves for both state estimation and updating model steps. For the state estimation, we just consider B_t as the bounding box of all positive pixels of $\mathbf{B}$. In that way, the whole state is estimated in one step, avoiding us some more consuming operations (GHT computed with several scales for example). For the updating step, let us denote $\mu_c = 0.90$ a color updating coefficient, H_t^B the color histogram of $\mathbf{I}_t|_{B_t}$. Then, we define H_t, the model color histogram by:

$$H_t = \mu_c \cdot H_{t-1} + (1 - \mu_c) \cdot H_t^B \tag{3.13}$$

To update the R-Table, we consider r the centroid of $\mathbf{B}|_{B_t}$, and an updating rate μ_g ($\mu_g = \mu_c$). Then, for all pixels p in B_t for which $\mathbf{M}(p) > \epsilon_M$, (i.e

gradient magnitude above a threshold), we update $R(\theta_p)$ as follows:

- if $\vec{u} = \overrightarrow{pr}$ is not in $R(\theta_p)$, we add an entry $(\vec{u}, \omega_{\vec{u}} = \mathbf{M}(p))$
- otherwise, we update $\omega_{\vec{u}}$ as follows:

$$\omega_{\vec{u}} \leftarrow \frac{\mu_g \cdot \omega_{\vec{u}} + (1 - \mu_g) \cdot \mathbf{B}(p) \cdot \mathbf{M}(p)}{2} \tag{3.14}$$

The aim is to reinforce the weights of displacements already present in the R-Table: if some pixels have correctly voted previously, we suppose that they are more prone to belong to the object. We then sort all displacements, in each entry $R(\theta_p)$, according to their weights, and keep only the $N_g = 50$ strongest displacements. If we compare our tracker with other

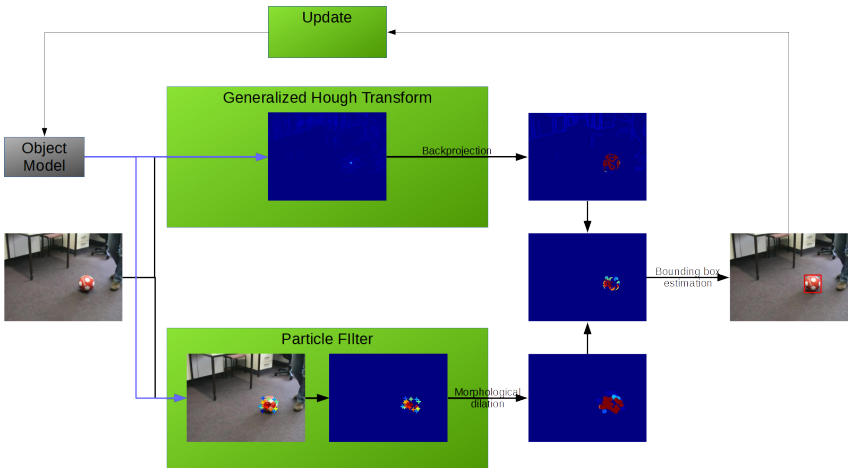


FIGURE 3.7: Diagram of tracker [TM15]

Hough-based ones, PixelTrack [DG13] is the closest one to ours. The differences between the two trackers can be detailed as follows:

- Our Hough-part tracker is closer to the original method [Bal81]. Indeed, our R-Table is indexed only by gradient orientation, while Duffner indexes it by using both gradient orientation and HSV pixels colors. Our aim was to consider the color and the shape parts as independently as possible, by exploiting them with different algorithms: the Particle Filter and the GHT respectively. Duffner's solution has the advantage to allow the R-Table to integrate more information, while maintaining the simplicity of the R-Table. However, Duffner's R-Table is more sensitive to noise, as entries are dependent on Hue and saturation values.

- Duffner proceeds to a segmentation using pixel colors, and requires a model for the foreground and the background (under the form of color histograms). His segmentation also requires a Bayesian model. In our case, we only use a model color histogram, combined with a particle filter

- Duffner's position estimation is computed from two outputs: one obtained by the GHT, one by computing the centroid of the color segmentation map. In our case, we are estimating the whole state (bounding box's center, size and orientation), by estimating the bounding box of the binarized fusion map $\mathbf{B}_t$

However, the two approaches are very close in terms of feature space (low-level features in both cases), the important role of the GHT, and a step to quickly compute a segmentation map in order to update target's models.

In this section, we presented the first version of our tracker, combining GHT and color histogram features. The aim was to maintain the use of low-level features, and to propose a very light tracker. Indeed, we kept important steps of tracking (model initialization and update, and tracking itself), but we discarded some usable information (target motion or other dynamical model). Finally, the presented tracker has several flaws, that we now discuss, together with some solutions.

3.3.3 Transitional tracker

The aim of the previous tracker was to propose a very simple algorithm, and so we discarded some classical elements used in tracking. We have fulfilled our goal to work on low-level features, as our tracker is exclusively based on gradient and color features. However, some important elements for tracking are missing.

The lack of a "real" prediction model (ours is just defined by the way to draw particles) is very important since the presented tracker is mainly based on the GHT, and some abrupt transformations can imply a failure of the GHT (The third picture from Fig. 3.8 illustrates it).

The second issue concerns the computation of the fusion map $\mathbf{B}$ (see Eq. 3.12), used for scale estimation. Before the threshold operation, for a pixel p, $\mathbf{B}(p)$ is strictly positive when both $\mathbf{BP}(p)$ and $\delta_{\mathbf{SE}}(\mathbf{OP}(p))$ are strictly positive (product). It means that a pixel with a positive value in $\mathbf{B}$ must have voted in the GHT process ($\mathbf{BP}_t(p) > 0$), and be spatially

close to one particle ($(\delta_{\mathbf{SE}}(\mathbf{OP}_t))(p) > 0$). Fulfilling these conditions can be hard (it depends on the size of the particle set, the number of displacements stored in the R-Table and on the target size) and the quantity of pixels with positive values in $\mathbf{B}$ is decreasing over the time, leading to a wrong size estimation. Moreover, for model update, with the previous method, only positive pixels in $\mathbf{BP}$ can participate to the R-Table update.

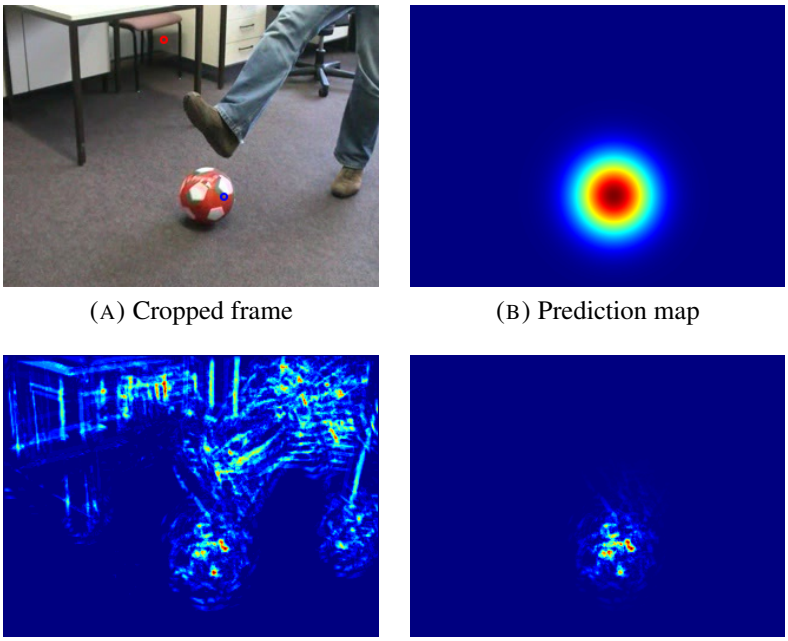
We addressed these two issues as follows:

- For the prediction model, given B_{t-1}, the previous estimated state, we define a normalized isotropic Gaussian map $\mathbf{PR}_{t-1}$ centered in c_{t-1}, and of standard deviation $\sigma_0 \cdot \min(w_{t-1}, h_{t-1})$, with $\sigma_0 = 0.70$. $\mathbf{PR}_{t-1}$ represents a prediction of the position, given the last estimated one, as high values in this map are for pixels close to c_{t-1}. This prediction model is fast to compute, and has been used in the literature [PMB15]. It is also possible to use a richer prediction model, by considering the motion of the target, as Breitenstein [Bre+09] proposed. Fig. 3.8 illustrates the usefulness of a prediction map, on a cropped frame from *ball* sequence. On the first line, one cropped frame is displayed, the red circle corresponds to the peak of the GHT (image below), while the blue one corresponds to the peak obtained from the per-pixel product between the GHT and the prediction map (on the right). Then, by defining the map $\mathbf{PHT}_t = \mathbf{HT}_t \cdot \mathbf{PR}_{t-1}$, the position of the target is determined by:

$$c_t = \underset{x}{\operatorname{argmax}}\, f_{\mathbf{PHT}_t}(x) \tag{3.15}$$

  where $f_{\mathbf{PHT}_t}$ is the integral of $\mathbf{PHT}_t$ in the rectangle centered on x, and of size $(0.30 \cdot w_{t-1}, 0.30 \cdot h_{t-1})$. The goal of Eq. 3.15 is to add some robustness with regard to deformations to the GHT, by considering that a pixel voting correctly does no longer have to vote accurately for a peak, but for a pixel close to this peak. Compared to Eq. 2.29, the method modeled by Eq. 3.15 is more robust to scaling and small deformations.

- For the scale estimation issue, we replace the dilated sparse observation map $\delta_{\mathbf{SE}}(\mathbf{OP})$ by a dense observation color map. After estimating m_t, the maxima of $\mathbf{HT}_t$, we define a surrounding area S_t as a rectangle of size $(\beta \cdot w_{t-1}, \beta \cdot h_{t-1})$ ($\beta > 1$), and excluding the rectangle $\{m_t, w_{t-1}, h_{t-1}\}$. Fig. 3.9 illustrates one surrounding area. The blue part corresponds to S_t, the red one to the bounding



(A) Cropped frame (B) Prediction map

(C) GHT (D) Product of GHT and prediction map

FIGURE 3.8: Impact of prediction map in the GHT

box (m_t, w_{t-1}, h_{t-1}). Let us define H_t^{bck} the normalized histogram

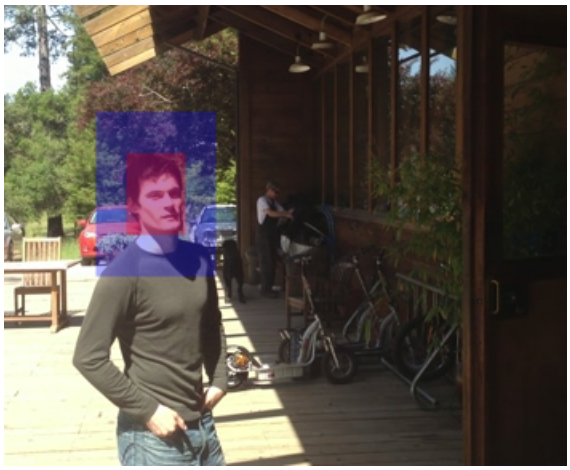


FIGURE 3.9: Surrounding area from *sunshade*.

of $I(S_t)$. Then, for each bin i, we define a ratio histogram H_t^R such that:

$$H_t^R(i) = \min(\frac{H_{t-1}(i)}{H_t^{bck}(i)}, 1.0) \tag{3.16}$$

and then build the map of the quantized color image of $\mathbf{I}_t$ using

H_t^R. This formulation differs from our approach in [TM15]. Indeed, while our previous work was based on Bhattacharyya coefficient (Eq. 2.7), this one aims to measure how likely one color belongs to the target or to the background: the more one color is present in the target and rare in the background, the higher H_t^R is. However, for color as much present in the foreground as in the background, Eq 3.16 will be close to 1.0. Let us call $\mathbf{C}_t = H_t^R(\mathbf{I}_t(p))$ the obtained map. Fig. 3.10 illustrates an area from one frame of the *sphere* sequence mapped using a RGB histogram of size $8 \times 8 \times 8$. Blue corresponds to low values, red to high ones. The red area corresponds to the foreground, the blue one to the background. Finally,

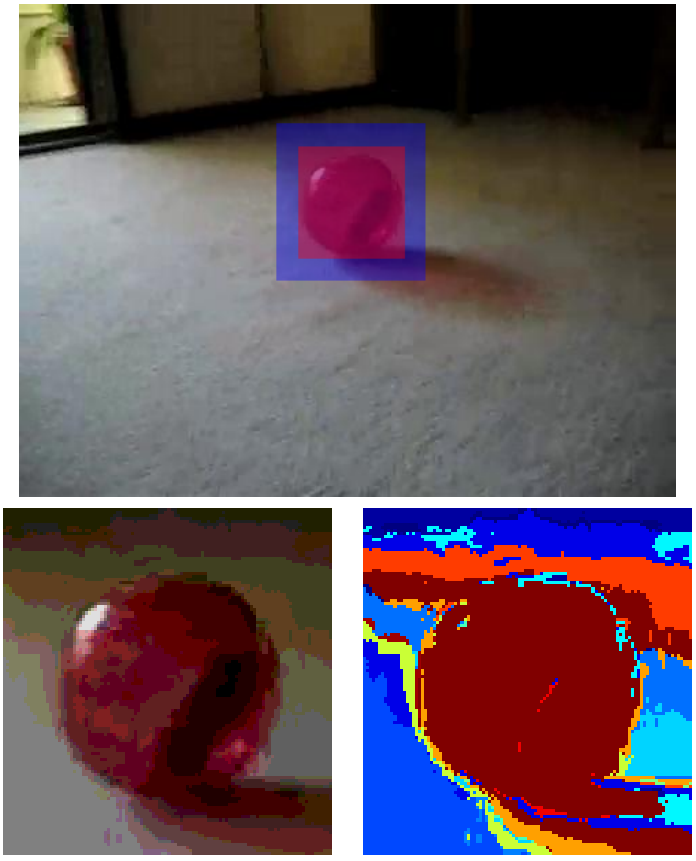


FIGURE 3.10: Mapping of the blue and red areas using H_t^R (Eq 3.16)

$\mathbf{B}_t$ is obtained by taking the bounding box of:

$$\{p|(\mathbf{B}(p) > \epsilon_B) \wedge (\mathbf{C}_t(p) > \epsilon_C)\} \tag{3.17}$$

where ϵ_B and ϵ_C are two thresholds. Like [TM15], scale estimation is still dependent on the realization of two conditions. But, as $\mathbf{C}_t$

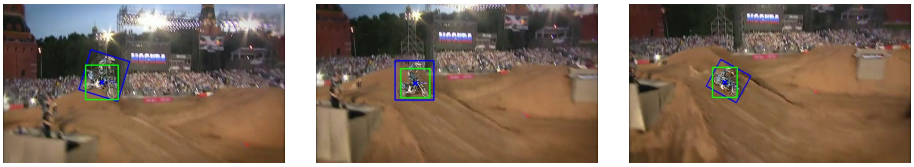


FIGURE 3.11: Some non consecutive frames from *motocross* sequence

is denser than $\mathbf{OP}_t$, the cardinality of the set of pixels defined by Eq. 3.17 if higher than previously. For the updating process, we also changed the method, by considering that a pixel $p \in B_t$ belongs to the target if:

$$\frac{\mathbf{BP}_t(p) + \mathbf{C}_t(p)}{2} > \epsilon_{fgd} \tag{3.18}$$

with $\epsilon_{fgd} = 0.45$. This condition, less restrictive than before, leads to a higher quantity of pixels used for the R-Table update. Fig. 3.11 illustrates a correct tracking for one moving target

One last weakness of [TM15] concerns the use of the particle filter: as we choose to be as simple as possible (for speed purpose), the resampling and propagation steps are used in their simplest form (full resampling according to a 2D Gaussian for the first, dynamical model defined with the Identity Matrix for the second) which does not let us take benefits from the particle filter (coping with dynamical model with the propagation, removing wrong hypothetical states and keep correct ones with the resampling).

This tracker performs better than [TM15], and is able to track accurately in more sequences (see Fig. 3.11). However, it only partially solves problems of the previous one, and still has several flaws.

3.3.4 Final tracker

The tracker presented previously provides decent results, but still suffers from flaws:

- The color-part has a secondary role compared to the GHT: the background histogram H_t^R is built according to m_t, the maxima of $\mathbf{HT}_t$. Conceptually, this issue is severe, as the role of the color model is only to refine the state estimation. For our work, as we aim to study interests of low-level features, we prefer having a more balanced role of the two models

- Estimating the object dimension by the method proposed in the previous section has a tendency to diverge: with cluttered backgrounds, the condition defined by Eq. 3.17 may be fulfilled by background pixels, and even an isolated background pixel is sufficient to wrongly estimate object size.

  From [TM15] to enhancements proposed in Section. 3.3.3, for scale estimation, we did not simplify the scale estimation (Eq. 3.17), which is still based on a conjunction of two conditions. With high thresholds, very few pixels are fulfilling these two conditions, and then, the scale has a tendency to dramatically decrease

- For state estimation, we are using Eq. 3.17, which requires two thresholds. For model update, based on Eq. 3.18, we define another threshold ϵ_{fgd}. The two sets of pixels validating respectively Eq. 3.17 and Eq. 3.18 are different. Moreover, these two equations require three parameters to define, complexifying the optimization of the algorithm

- Eq. 3.15 is used to estimate object position. Even though this operation can be quickly computed by integral image, it is still more costly than a simple argmax operation, and is dependent of the size of the window. We are then loosing some flexibility compared to [TM15] as we add one parameter

All these reasons led us to improve the tracker. In this section, we propose some corrections to get an effective tracker [TM17], that will be used for the whole experiments presented in Section. 3.4. For reading ease, all numerical values of the parameters will not be presented in this part. We also decided to split details into 3 parts:

1. We will explain modifications made for position estimation. At this level, the tracker is already giving interesting results

2. We will explain how to generate a segmentation map, used for scale and orientation updating

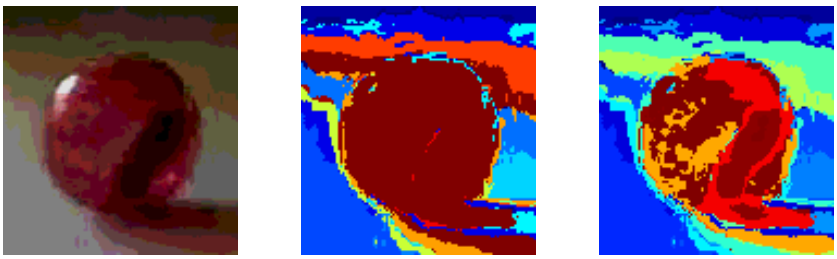
3. Details about model update will be provided

3.3.4.1 Position estimation

First, to remove the dependency of the color part to the GHT, we define the surrounding area S_t as the rectangle $\{c_{t-1}, \alpha \cdot w_{t-1}, \alpha \cdot h_{t-1}\}$, excluding $R_t = \{c_{t-1}, w_{t-1}, h_{t-1}\}$ (α will be the surrounding area coefficient). By centering the surrounding area at c_{t-1} instead of m_t (Hough peak), the foreground/background areas are no longer dependent of the GHT. The second modification concerns Eq. 3.16, used to classify a pixel as foreground or not. For all pixels $p \in (S_t \cup R_t)$, we follow Possegger's formulation [PMB15] by defining a new measure of confidence $\mathbf{F}_{\mathcal{O}}^{(H_{t-1}, H_t^{bck})}$ such that, by denoting q_t^p the quantified color of $\mathbf{I}_t(p)$:

$$\mathbf{F}_{\mathcal{O}}^{(H_{t-1}, H_t^{bck})}(q_t^p) = \begin{cases} \frac{H_{t-1}(q_t^p)}{H_{t-1}(q_t^p) + H_t^{bck}(q_t^p)} & \textit{if } p \in (S_t \cup B_{t-1}) \\ 0 & \textit{otherwise} \end{cases} \tag{3.19}$$

Compared to Eq. 3.16, there is no risk of division by 0 for pixels inside $(S_t \cup R_t)$ (denominator always strictly positive). By extension, $\mathbf{F}_{\mathcal{O}}^{(H_{t-1}, H_t^{bck})}(q_t^p) \leq 1.0$. Moreover, for colors such as $H_{t-1}(i) = H_t^{bck}(i)$, while Eq. 3.16 will give 1.0, Eq. 3.19 will give 0.5, reducing impact of colors as much present in the foreground as in the background. If we reconsider the example from *sphere* sequence, Fig. 3.12 illustrates improvements induced by Eq. 3.19: pixels from the shadow now appear in green, while before, it appeared in red. This formulation comes from a Bayesian formulation of the likelihood, and we refer to [PMB15] for a detailed explanation. From this new function, we can now define the method to estimate the object center at t,



(A) Quantized image (B) Using Eq. 3.16 (C) Using Eq. 3.19

FIGURE 3.12: Impact of Possegger's formulation [PMB15] on the color-based confidence map

given H_{t-1} and B_{t-1}. Let us define $\mathbf{S}_{c,t}$ such that:

$$\mathbf{S}_{c,t}(B_x) = \begin{cases} \frac{\sum_{p \in B_x} \mathbf{F}_{\mathcal{O}}^{(H_{t-1}, H_t^{bck})}(p)}{w_{t-1} \cdot h_{t-1}} & \textit{if } x \in (B_{t-1} \cup S_t) \\ 0 & \textit{otherwise} \end{cases} \tag{3.20}$$

where $B_x = \{x, w_{t-1}, h_{t-1}\}$.

Second, the GHT is computed as in [TM15] (Eq. 3.10, each displacement casts a vote given by its weight). However, we are blurring the obtained Hough Transform (with a 3×3 Gaussian Kernel), and denote by $\mathcal{GB}(\mathbf{HT}_t)$ the output. This operation, lighter than computing $f_{\mathbf{PHT}}$ in Eq. 3.15, serves to add robustness to scaling and deformations.

Finally, for position estimation, we add a prediction model $\mathbf{PR}_t$ defined as previously. Thus, to estimate object's center, we consider $\mathbf{M}_t = \mathcal{GB}(\mathbf{HT}_t) \cdot \mathbf{S}_{c,t} \cdot \mathbf{PR}_t$, and estimate c_t by:

$$c_t = \begin{cases} \text{argmax}_x(\mathbf{M}_t(x)) & \textit{if } \max_x \mathbf{M}_t(x) \neq 0 \\ c_{t-1} + \overrightarrow{c_{t-2}c_{t-1}} & \textit{otherwise} \end{cases} \tag{3.21}$$

The second case of the equation is important as, if, for all x, $\mathbf{M}_t(x) = 0$, we can not determine an object center, and then, decide to estimate it based on pure prediction, given the two last estimated states. At this point, the tracker, combined with the updating process detailed further, is already giving decent results, (details Section. 3.4). Compared to the literature:

- Compared to Duffner's PixelTrack [DG13], who also proceeds to a light color segmentation, we merge the outputs of the two trackers earlier: Duffner computes one partial object's center from each tracker (GHT's peak and centroid of the color confidence map), then estimate object's center with a linear combination of the two. In our case, we merge the two center confidence maps ($\mathcal{GB}(\mathbf{HT}_t)$ and $\mathbf{S}_{c,t}$), then estimate object center according to Eq. 3.21. The weight of each partial center is also dependent on the amplitude of the peak of the GHT. Moreover, Duffner' segmentation process requires foreground and background stored models and use a Bayesian formulation, while ours only need object's color histogram (the background one is computed at each frame)

- Bertinetto's STAPLE [Ber+16] proceeds in a way similar to ours: after estimating two confidence maps (one from a correlation-filer,

the second from color histogram), he merges the two maps with a linear combination. In that way, he has to consider difference of amplitude between the two confidence maps, while in our case, the issue is related to disjoint supports of $\mathbf{HT}_t$ and $\mathbf{S}_{c,t}$ (solved by the second case of Eq. 3.21)

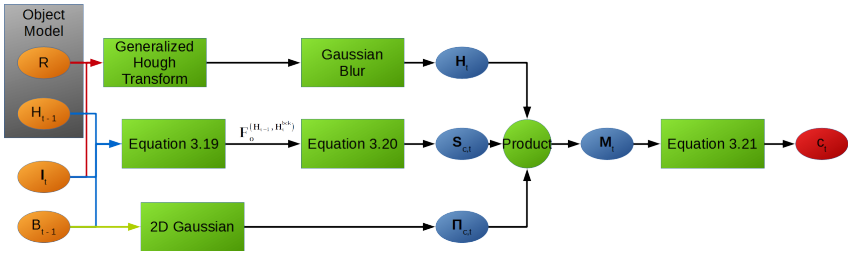


FIGURE 3.13: Diagram of position estimation (better in color)

The diagram of Fig. 3.13 illustrates all steps for position estimation. All three entries (in blue) serving as an input for the product function can be computed in parallel.

3.3.4.2 Scale and orientation estimations

Previously, we explained modifications made in order to estimate object's position. Now, we explain how to estimate both scale and orientation. The aim of this step is to build a confidence map, that will serve for tracking and also for updating task.

To do that, we will consider two confidence maps (one for each feature). The first one, from the color model, is the map $\mathbf{F}_{\mathcal{O}}^{(H_{t-1},H_t^{bck})}$ defined by Eq. 3.19. The second one, coming from the GHT, is another version of the backprojection map $\mathbf{BP}_t$:

$$\mathbf{BP}_t(p) = \frac{\sum\limits_{(\overrightarrow{u},\omega)\in R(\theta_p)} \mathbf{M}_t(p+\overrightarrow{u})}{|R(\theta_p)|} \tag{3.22}$$

with $|R(\theta_p)|$ the cardinality of $R(\theta_p)$.

Then, by denoting $\mathbf{BF}_t = 0.5 \cdot (\mathbf{BP}_t + \mathbf{F}_{\mathcal{O}}^{(H_{t-1},H_t^{bck})})$, and, inspired by Possegger's works [PMB15], we define the set of pixels that belongs to the object as $OP_t = \{p|\mathbf{BF}_t(p) > \epsilon_{\mathbf{BF}}\} \cup \{c_t, \beta \cdot w_{t-1}, \beta \cdot h_{t-1}, \theta_{t-1}\}$, the rectangle $\{c_t, \beta \cdot w_{t-1}, \beta \cdot h_{t-1}, \theta_{t-1}\}$ being a safe foreground area (based on the assumption that all pixels close to c_t are more prone to belong to the object), and β the safe foreground area coefficient. After that, we compute

a connected component analysis within OP_t, to get all pixels connected to c_{t-1}, and discard isolated ones, as they can lead to a scale overestimation. Finally, the last step is the estimation of the oriented bounding box $\hat{B}_t = \{\hat{c}_t, \hat{w}_t, \hat{h}_t, \hat{\theta}_t\}$ of all these connected pixels (rotating caliper algorithm implementation from OpenCV).

Given the last estimated state B_{t-1}, if each estimated state component (except c_{t-1}) is too far from the last estimated one, we do not change it. Otherwise, we update it smoothly:

$$X_t = (1 - \mu) \cdot X_{t-1} + \mu \cdot \hat{X}_t \tag{3.23}$$

with $X \in \{w, h, \theta\}$.

This version differs from [TM17], which only proceeds to a scale estimation (estimation to 4 parameters instead of 2). To do so, after estimating $\hat{B}_t$ (in this case, we are working with bounding boxes without orientation), we calculate relative area variation between B_{t-1} and $\hat{B}_t$. If area variation is above $\epsilon_{\mathcal{A}}$, we do not update scales. Otherwise:

$$X_t = \lambda_t \cdot X_{t-1} \tag{3.24}$$

with $X \in \{w, h\}$ and $\lambda_t = \min(1 + \lambda_A, \max(1 - \lambda_A, \frac{\mathcal{A}(\hat{B}_t)}{\mathcal{A}(B_{t-1})}))$, and λ_A the max area variation. The remaining step is the model update.

3.3.4.3 Updating model

For the updating process, we use the merged confidence map $\mathbf{BF}_t$ as a support for updating. We follow our previous work [TM15] by smoothly updating the color histogram, still using Eq. 3.13. For the R-Table, we apply a more sophisticated updating process than in [TM15]. Indeed, the first step is to weaken weights of all displacements already present in the R-Table R:

$$\forall \theta \in \{0, 1, \ldots, n_c - 1\}, \forall (\vec{u}, \omega) \in R(\theta), \omega \leftarrow (1 - \mu_g) \cdot \omega \tag{3.25}$$

Then, we add new displacements, or update already existing ones in a similar way as Section. 3.3.2. However, for new displacements, the new weight is equal to $\mu_g \cdot \mathbf{BF}_t$. For already existing ones, we increment their weights using the same weight $\mu_g \cdot \mathbf{BF}_t$. The weight remains then inside

$[0, 1]$. In that way, compared to Eq. 3.14 which requires gradient magnitude, we no longer have to consider the dynamic of the gradient magnitude. The interest of combining the weakening mechanism (Eq. 3.25) and the weight increases is that the role of the first one is to reduce influence of all displacements, in particular those from the background, whereas the second step compensates the weakening.

3.3.4.4 Conclusion

We presented three different trackers combining a shape-based model and a color-based one. First, we described a tracker combining the GHT, exploited in its simplest form, which is common to all three methods, and a Particle Filter, applied in a very simple form and based on color histogram. The two main issues of this tracker was the lack of real prediction model and the dependence of the Particle Filter toward the GHT. Second, we discarded the Particle Filter part to use only a color histogram as a feature to classify pixels as background or foreground, and add a 2D Gaussian acting as a simple prediction model. Third, in Section. 3.3.4, inspired by Possegger's background/foreground discrimination, we changed the formula to establish the color confidence map, providing a better segmentation (see Fig. 3.12). We also removed the dependency of the color model to the GHT, by making it dependent on the last estimated state. Then, the two methods, combined with a prediction model, are merged (with a pixel-wise multiplication) to produce a map, that can be already used for tracking. Finally, we extract two confidence maps (one for each feature), serving as a base for the complete state estimation.

These three proposed trackers have two common points: one unique algorithm exploiting gradient features (the GHT), and the exclusive use of local and low-level features (gradient and color histogram). Compared to Hough-based trackers presented in Section. 3.2.2, the GHT is used in its simplest form: computed by gradient orientation (as opposed to keypoint features for example) and a codebook in the form of a simple array indexed by gradient orientation (compared to random ferns structures). In every cases, our GHT is identical to the original GHT [Bal81], and is used without extensions provided to cope with scale or orientation changes (votes in a 4D parameters space).

In the final section, we will validate the last proposed tracker by testing and evaluating it on academic datasets. We tested only [TM17]. In

terms of performances, neither of the two others trackers is competitive: the lack of prediction model is critical for [TM15], while scale overestimation of the transitional tracker leads to low accuracy. In terms of speed, we can suppose that the transitional tracker and [TM17] have similar performance. However, for [TM15], the particle filter part can be very costly, compared to the computation of Eq. 3.19, as it requires histogram and Bhattacharyya coefficient computations. Even though these two operations can be optimized, with a high number of particles, they are more computationally expensive than Eq. 3.19.

3.4 Results

We present experiments made with our tracker described in Section. 3.3.4 on academic datasets [Kri+13]; [KPL+]; [Kri+15]. Before that, as our tracker is running above real-time, we consider necessary to detail our implementation. Then, we focus on some experiments. Academic datasets first serve as a support for parameters tuning. Then, we study the influence of changing the feature spaces.

3.4.1 Implementation details

3.4.1.1 Optimization

All our work has been implemented in C++, using OpenCV 2.4.9 [2]. Our final tracker was tested on a laptop machine, equipped with an Intel Core i7-4700HQ, at 2.40 GHz, and coded for a single thread processing.

It terms of implementation, pixel access is done using image pointers. For optimization purpose, at frame $t+1$, given $R_t = \{x_t^R, y_t^R, w_t^R, h_t^R\}$ the rectangle enclosing B_t and parallel to x- and y- axes, the tracking process is done in an area defined by $(c_t, \gamma \cdot w_t^R, \gamma \cdot h_t^R)$ ($\gamma = \sqrt{2}$ is the search window coefficient). The aim is to reduce the search window to a rectangle of area $2 \cdot w_t \cdot h_t$ which contains B_t (for any orientation of B_t).

We compute gradient by using a 5×5 discrete Gaussian kernel, with $\sigma = 1$. For histogram computation, we use a LookUp Table (LUT) LUT_c. Given a defined size of histogram n_c, and the range of one octet ($\{0, 1, \ldots, 255\}$), we define the width of one bin $\delta_c = \frac{256}{n_c}$, and then create

[2] http://opencv.org/

LUT_c such that:

$$\forall i \in [0, 255], LUT_c(i) = \lfloor \frac{i}{\delta_c} \rfloor \tag{3.26}$$

with $\lfloor \bullet \rfloor$ the floor operator. This operation is computed offline. Computation histogram for LUT is faster than the naive implementation, as all the division operations are done offline (during the LUT construction), while online, computation only requires to access elements.

To compute $S_{c,t}$, defined by Eq. 3.20, we compute integral images [Lew95]. Given an image I of size (w, h), its integral image $\mathcal{I}(\mathbf{I})$ is defined by:

$$\forall (x, y) \in \{1, \ldots, w\} \times \{1, \ldots, h\}, \mathcal{I}(\mathbf{I})(x, y) = \sum_{(i,j) \in [0,x-1] \times [0,y-1]} \mathbf{I}(i, j) \tag{3.27}$$

In that case, given a rectangle $R = \{c_0, w_0, h_0\}$ (with $c_0 = (x_0, y_0)$), the sum of all pixels values of $\mathbf{I}_{|R}$ can be computed with 4 array accesses and three additions:

$$\begin{aligned} \sum_{p \in R} \mathbf{I}(p) = \mathcal{I}(\mathbf{I})(x_0 + w_0, y_0 + h_0) + \mathcal{I}(\mathbf{I})(x_0, y_0) - \\ \mathcal{I}(\mathbf{I})(x_0 + w_0, y_0) - \mathcal{I}(\mathbf{I})(x_0, y_0 + h_0) \end{aligned} \tag{3.28}$$

Even though we aim to estimate object's orientation too, we evaluate $S_{c,t}$ with bounding boxes parallel to x and y axes, to leverage integral images.

One last algorithmic optimization concerns the R-Table update. To discard weak displacements, we first sort them according to their weights. Then, when we need to update the weight of an already existing displacement, at each gradient orientation, we sort displacements according to their ascending abscissas. Finally, to find the existence of a displacement, given the index θ and the displacement $\overrightarrow{u} = (x, y)$ to be inserted into $R(\theta)$, as soon as in $R(\theta)$ we find a displacement with an abscissa greater than x, we are sure that $\overrightarrow{u}$ does not exist in $R(\theta)$ yet.

Otherwise, we did not proceed to other algorithmic optimization: in Figure. 3.13, the three boxes *Generalized Hough Transform*, *Equation 2* and *2D Gaussian* are done independently (and then, with 3 *for* loop), while these three loops can merge into one unique, potentially providing us a faster algorithm (in single thread implementation only, in multi-thread, these loops can be computed in parallel). The computation of the prediction map $\mathbf{PR}_t$ has been done exactly, while it can be computed (with

a certain approximation) using a LUT. This method is all the more critical that for some sequences involving big objects, this operation is very costly.

Finally, our tracker has been optimized without approximation, or loss of accuracy. However, at this point, it has proven to be fast (above real-time), accurate and robust. In the following of this section, we will present the evaluation.

3.4.2 VOT datasets

3.4.2.1 History of the VOT Challenge

We present the VOT challenge in this section, and the main criterion used to test end evaluate trackers. This will be important not only for results on the dataset, but also for an understanding of the parameter tuning detailed in Section. 3.4.2.2.

Since 2013 [Kri+13], the VOT committee proposes annually a video dataset to test and compare trackers. They also propose the TraX toolkit [Čeh17] as a standard to run and evaluate trackers. For our experiments, we use a deprecated version of the toolkit, giving the same results as the recent one, but less flexible.

Each video is annotated by the ground truth, and each frame is annotated according to its specific difficulties:

- Occlusion (Occ)
- Illumination change (Ill)
- Motion change (Mot)
- Size change (Siz)
- Camera motion (Cam)
- Empty (no difficulty; Emp)

Then, to evaluate trackers, the committee considers three criteria:

- *Accuracy*, based on overlap measure O: at a frame t, given the ground truth bounding box GT_t and an estimated bounding box B_t, the overlap measure, $O(GT_t, B_t)$ is given by:

$$O(GT_t, B_t) = \frac{GT_t \cap B_t}{GT_t \cup B_t} \tag{3.29}$$

Fig. 3.14 illustrates the overlapping measure: the blue rectangle is

FIGURE 3.14: Illustration of overlappping

the ground truth, the red one the estimated one. This measure is valued between 0 and 1

- *Robustness to failure*. A failure happens when overlap measure is equal to 0. In order to limit the penalty caused by the failure, each time a tracker fails, it is reinitialized 5 frames later. Accuracy measures are made only 10 frames after the failure

- *Speed*. In VOT2013 [Kri+13], speed was given in frame per second (with hardware characteristics). In VOT2014 [KPL+], a normalized speed was introduced. Before testing one tracker on the whole dataset, the toolkit measures the time t_δ necessary to compute a maximum (morphological dilation) filter of size 30×30 on a grayscale image of size 600×600. Then, the normalized speed, called Equivalent Filter Operation (EFO), is given by $\frac{t_t}{t_\delta}$, where t_t is the time to run the tracker on the whole dataset.

Each tracker is ranked in terms of accuracy and robustness. Two rankings are available. For the first one, the pooled rank, all sequences are concatenated, and average accuracy and robustness are computed for all frames. Trackers are then ranked according to average overlap (accuracy rank) and number of failure (robustness rank). For the second, the weighted rank, partial ranks are computed by ranking according to difficulties (for each difficulty, ranking is done in terms of accuracy and robustness too), and final ranking is done by averaging ranks. To remove dataset bias, the average is done by weighting all partial ranks: very frequent difficulties have

lower weights. In both cases, the AR rank is plotted (giving the so-called AR plot), with the average rank in abscissa and the robustness rank in ordinate. Statistics methods are also used to differentiate (or not) the rank of trackers by estimating whether their difference in accuracy or robustness is statistically significant (or not) (see [Kri+16a] for details). Since VOT2014 [KPL+], the minimal requirement to be coauthor in the competition report is to outperform the reference tracker NCC [BH01] in the challenge.

In VOT2015, the committee proposes a more interpretable criterion called *expected overlap*. It serves to rank competitors with one unique score, compared to the AR rank in VOT2014. Given a sequence s, a beginning frame index $t_i(s)$ and an ending frame index $t_e(s)$, the average overlap of the sequence $\phi(s)$ is equal to:

$$\phi(s) = \frac{1}{t_e(s) - t_i(s)} \sum_{t=t_i(s)}^{t_e(s)} O(GT_t, B_t) \tag{3.30}$$

And then, the expected overlap is equal to the average of the average overlap of all sequences. Even though this score has been proposed for VOT15, the version of the used toolkit also provides this result for VOT14.

The evaluation process from VOT challenge serves for the next section, in which we will detail parameters setting.

3.4.2.2 Parameter details

Selecting the best parameter set of an algorithm is a complicated task. In tracking context, estimating performances of one tracker and comparing it to another tracker is not a trivial task [Kri+16a]; [vLK16]. Moreover, our algorithm requires several parameters to set, and optimizing the whole set is a complex task.

Then, we distinguished only 4 parameters that can highly impact performances of the tracker: the size of the color histogram n_c, the number of index for the R-Table n_g, and the two updating rates μ_c and μ_g. The number of displacements in the R-Table N_g also has an important role, but secondary compared to n_g. Moreover, its optimal value is more related to the object size (the bigger it is, the higher N_g should be) than n_g, which is more related to the object shape. Then, we select the VOT2015 dataset [Kri+15] and the associated method of evaluation as a dataset of reference for tuning. Now, in terms of criterion to optimize, we can use

either the expected overlap or the AR-plot (weighted ranking, less biased). Even though the first one seems the most important one (as it serves to establish the global ranking on VOT2015) and choosing this criterion would simplify the task, we do not want to restrict to this, as accuracy and robustness can be more important depending on the task. As we aim to propose a reactive algorithm, time consumption is also a criterion we would like to optimize. Optimization has been done with the position tracker only (by discarding Section. 3.3.4.2) as using the whole algorithm would make the optimization task much more complex (more parameters to tune).

In that condition, the optimization problem consists in optimizing a function of $\mathbb{N} \times \mathbb{N} \times [0,1] \times [0,1]$ (the two integers correspond to n_c and n_g) to $\mathbb{R}^4$. Mathematically, to solve this optimization problem, we need to define a cost function to minimize. Then, we can proceed to a grid search in order to find a solution close to the optimal one, but the time necessary to test all combinations can be tremendous: if the grid is composed of 6 possible parameter values per parameter, there are $6^4 = 1296$ possible combinations, leading to a huge number of tests to execute. In our case, testing our tracker on VOT2015 takes 30 minutes. So, testing all 1296 combinations will take 648 hours (about 27 days). Then, we proceed by setting $n_c = 12$ and $n_g = 16$, and then, searching for couples (μ_c, μ_o) using a grid search. We tested different couples (n_c, n_g) and chose this one as it already gave correct performances (see [TM17]). Then, after choosing the best couple, we will deal will optimization of (n_c, n_g) by reiterating the same process, with (μ_c, μ_g) fixed, and (n_c, n_g) varying. Both parameters (μ_c, μ_g) are selected among $\{0, 0.2, 0.4, 0.6, 0.8, 1.0\}$, giving 36 tests to compute.

We show average overlap for all tests Fig. 3.15. Red points correspond to results obtained experimentally, while other points are obtained with bilinear interpolation. One first remark we have is that for null update rate of the R-Table ($\mu_g = 0$), expected overlap is very low, for any value of μ_c. The maximal relative variation between the lowest (0.1894 for $\mu_g = \mu_c = 0.0$) and the highest expected overlap (0.2668 for $\mu_g = 0.08$ and $\mu_c = 0.06$) is equal to 29%. Otherwise, for $\mu_g > 0$, and any value μ_c, expected overlap is close to the computed maximum. The relative variation is lower: with $(\mu_g, \mu_c) = (0.04, 0.00)$, expected overlap is equal to 0.2358, and relative difference is then equal to 10%. We can already discard all couples with $\mu_g = 0$. Fig. 3.16 and Fig. 3.17 illustrate accuracy and robustness ranks respectively (the smaller, the better). Fig. 3.16 shows

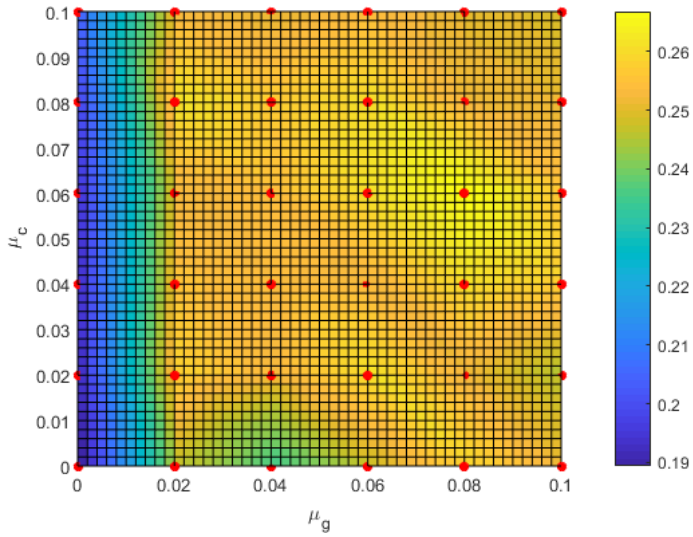


FIGURE 3.15: Expected overlap for different values for μ_c and μ_g

that all trackers have similar ranking (no significant difference): this criterion will not serve to differentiate parameters set. On the other hand, we can notice that Fig. 3.17 and Fig. 3.15 have inverted trends (and the range of ranking is much more dynamic too): the smaller the rank is, the higher the expected overlap is. Tab. 3.1 shows robustness ranking according to different difficulties. We can find that for $\mu_c = 0$, trackers perform badly in camera motion, empty and size change cases. Globally, as most trackers have strong ranks for illumination change, we can say that this difficulty is not relevant for our algorithm. For motion change, $\mu_g = 0.02$ performs very badly. Surprisingly, we can notice that $\mu_g = 0.04$ gives poor results for Occlusion issues. Finally, $\mu_g > 0.06$ gives globally good results in robustness. The last criterion is the computation time. As it is independent of update rates, we will not consider it for now.

Finally, if we want to select the best couple of results, two choices are possible:

- VOT15 [Kri+15] ranking is based on expected overlap. In this case, the best combination is $(\mu_g, \mu_c) = (0.08, 0.06)$
- The best ranking in terms of AR ranking is for $(\mu_g, \mu_c) = (0.10, 0.02)$. However, this combination leads to a low expected overlap compared to the best overlap found (0.2668 against 0.2453). The second best options are $(0.06, 0.06)$ and $(0.08, 0.02)$, which are close

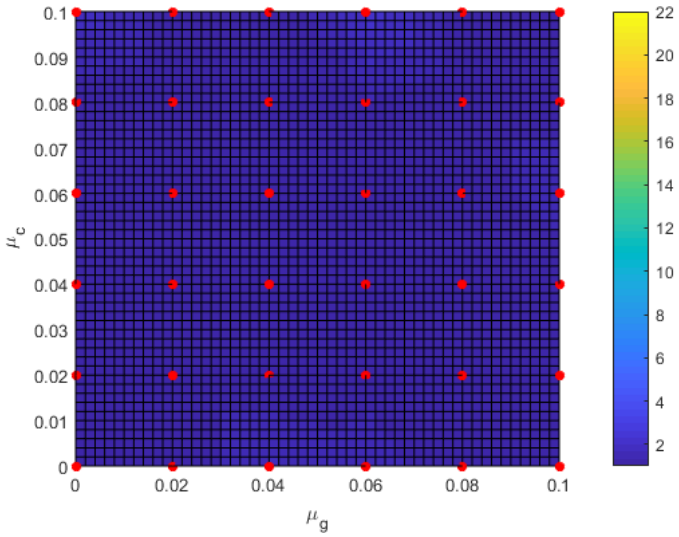


FIGURE 3.16: Accuracy rank for different values for μ_c and μ_g

to our first option. The first one gives expected overlap close to the maximum (0.2613), while the second is middle ranked (0.2518)

For the rest of the tests, we will choose $(\mu_g, \mu_c) = (0.08, 0.06)$ as the best combination.

Now, we need to tune the two parameters n_c and n_g. As for the two update rates, we proceed to a grid search by considering all combinations in $\{8, 12, 16\} \times \{8, 12, 16\}$. Expected overlap is displayed Fig. 3.18, also obtained with bilinear interpolation (only computed for integers values) showing no significant difference. As previously, accuracy rank does not vary much (9 trackers all ranked between the rank 1 and 2), and the same thing for the robustness (between 1 and 4). This suggests that the impact of both parameters we want to tune is limited, compared to (μ_g, μ_c). Speed is important in this case, and results are shown Fig. 3.21 (in fps). In terms of speed, all trackers are in the same order of magnitude, between 126.52 and 130.78 fps.

Finally, if we want to select the best couple, the couple $(n_c, n_g) = (12, 16)$ is the best ranked and the one with the highest expected overlap. We will then select this couple for experiments. This couple is the one we set at the beginning to initialize our optimization process. For our initialization $(n_c, n_g) = (12, 16)$, the couple $(\mu_g, \mu_c) = (0.08, 0.06)$ is the one optimizing results for VOT2015. Tab. 3.2 summarizes all parameters and chosen values. Parameters in bold are those that we optimize. We note that these values may not be the best ones for other datasets, due to the

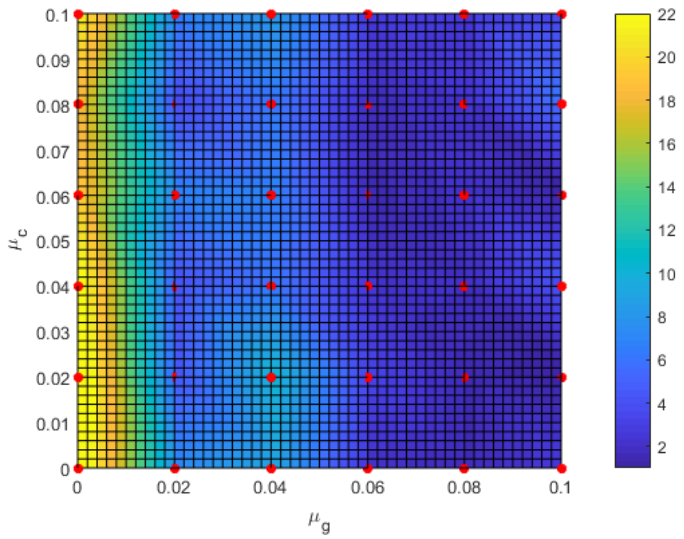


FIGURE 3.17: Robustness rank for different values for μ_c and μ_g

diversity of videos. Parameters shown in Tab. 3.2 are used in both VOT14 and VOT15. Until the end of the chapter, we will denote as CHT the position tracker only, CHTs the tracker estimating the position and the scale (aspect-ratio fixed), and CHTf the full one. As VOT2014 and VOT2015 proposed oriented bounding box ground truth, CHT and CHTs will be initialized with the smallest rectangle parallel to x- and y- axes enclosing the initial ground truth. The two next sections will be dedicated to results on VOT2014 and 2015 datasets, to position our work with regard to the state-of-the-art. Finally, we will propose an analysis of our algorithm.

3.4.2.3 VOT2014

VOT2014 [KPL+] is composed of 25 sequences taken from several datasets [Kri+13]; [WLY13]; [Sme+14]. Objects are manually annotated by slanted bounding boxes, unlike [Kri+13]. For each tracker, experiments are done in two ways:

- A tracker is initialized by the first frame only (model-free hypothesis) using the ground truth. The tracker then runs through the whole sequence. This experiment is called *baseline* experiment

- To evaluate the sensitivity to initialization, the second experiment consists in testing the tracker initialized by a disturbed bounding box (shift in position, size and orientation), according to a uniform

(μ_g, μ_c)	Cam.	Emp.	Ill.	Mot.	Occ.	Siz.
$(0.00, 0.00)$	29.00	33.00	26.00	31.00	13.00	7.00
$(0.00, 0.02)$	29.00	33.00	10.00	31.00	13.00	25.00
$(0.00, 0.04)$	36.00	29.00	**2.00**	31.00	**1.00**	35.00
$(0.00, 0.06)$	21.00	29.00	**2.00**	31.00	**1.00**	29.00
$(0.00, 0.08)$	28.00	20.00	**2.00**	31.00	**1.00**	25.00
$(0.00, 0.10)$	29.00	27.00	**2.00**	34.00	**1.00**	35.00
$(0.02, 0.00)$	16.00	6.00	3.00	15.00	**1.00**	**2.00**
$(0.02, 0.02)$	**2.00**	**3.00**	13.00	5.00	**3.00**	**1.00**
$(0.02, 0.04)$	**2.00**	**1.00**	3.00	5.00	**3.00**	12.00
$(0.02, 0.06)$	7.00	**1.00**	3.00	15.00	13.00	**2.00**
$(0.02, 0.08)$	**2.00**	**1.00**	**2.00**	13.00	**3.00**	**2.00**
$(0.02, 0.10)$	6.00	**1.00**	3.00	16.00	13.00	**2.00**
$(0.04, 0.00)$	28.00	6.00	**2.00**	15.00	**1.00**	**2.00**
$(0.04, 0.02)$	**2.00**	**1.00**	26.00	**1.00**	30.00	**1.00**
$(0.04, 0.04)$	**2.00**	**1.00**	3.00	**1.00**	27.00	**2.00**
$(0.04, 0.06)$	3.00	**1.00**	**2.00**	**1.00**	31.00	**2.00**
$(0.04, 0.08)$	**2.00**	**1.00**	3.00	**1.00**	28.00	**2.00**
$(0.04, 0.10)$	16.00	**1.00**	3.00	4.00	24.00	**2.00**
$(0.06, 0.00)$	3.00	4.00	**2.00**	**1.00**	**1.00**	**2.00**
$(0.06, 0.02)$	6.00	**1.00**	3.00	4.00	**3.00**	**2.00**
$(0.06, 0.04)$	6.00	**1.00**	**2.00**	**1.00**	**3.00**	**1.00**
$(0.06, 0.06)$	3.00	**1.00**	**2.00**	**1.00**	**1.00**	**1.00**
$(0.06, 0.08)$	3.00	**1.00**	**2.00**	**1.00**	**3.00**	**1.00**
$(0.06, 0.10)$	6.00	**1.00**	3.00	**2.00**	**3.00**	**2.00**
$(0.08, 0.00)$	**2.00**	**3.00**	**2.00**	**1.00**	**1.00**	**1.00**
$(0.08, 0.02)$	**2.00**	**1.00**	**2.00**	**2.00**	**1.00**	**1.00**
$(0.08, 0.04)$	**1.00**	6.00	**1.00**	**1.00**	**1.00**	**2.00**
$(0.08, 0.06)$	**2.00**	**1.00**	**2.00**	4.00	**3.00**	**1.00**
$(0.08, 0.08)$	3.00	**1.00**	**2.00**	5.00	**1.00**	**2.00**
$(0.08, 0.10)$	3.00	**3.00**	**2.00**	5.00	**1.00**	**2.00**
$(0.10, 0.00)$	**1.00**	9.00	3.00	**1.00**	**1.00**	**1.00**
$(0.10, 0.02)$	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
$(0.10, 0.04)$	**1.00**	20.00	**1.00**	**1.00**	**1.00**	**1.00**
$(0.10, 0.06)$	**1.00**	**1.00**	**1.00**	**2.00**	**1.00**	**1.00**
$(0.10, 0.08)$	3.00	6.00	**2.00**	5.00	18.00	**2.00**
$(0.10, 0.10)$	**2.00**	9.00	**2.00**	4.00	**3.00**	**1.00**

TABLE 3.1: Robustness ranking for different couples (μ_g, μ_c)

random law of range equal to 10% of the original value, while orientation is perturbed with a uniform random process in $[-0.1\,\mathrm{rad}, +0.1\,\mathrm{rad}]$. This experiment is called *region noise* experiment. To reduce bias,

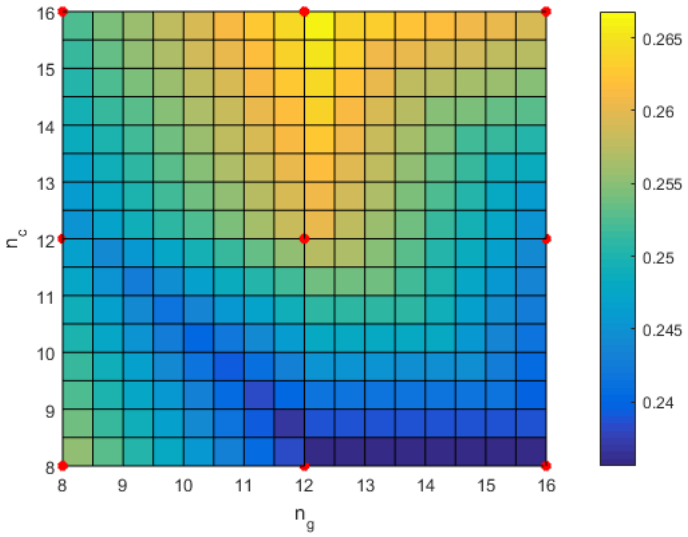


FIGURE 3.18: Expected overlap for different values of n_c and n_g

each sequence is computed 15 times, with different initial disturbance. Results are obtained by averaging the 15 measures

We compute evaluation and ranking for the whole set of competitors (the VOT committee provides results from 40 trackers). In VOT2014, trackers are ranked according to the AR rank, and in the AR plot, best trackers are on the upper-right part of the figure. However, we will only display results for some of them: DSST [Dan+14a], VOT2014's winner based on correlation filter, Hough-based trackers [DG13]; [GRB13]; [MP13] and real-time trackers [DG13]; [MP13]; [MP14]; [VM14]; [Lew95]; [Hen+15]. Tab. 3.3, 3.4, 3.5 and Fig. 3.24 summarize results for baseline, region noise and overall cases respectively. *Overall* results are obtained by averaging *baseline* and *region noise* results. Globally, the trends obtained for the baseline and the noisy tests are the same. We note that, in terms of ranking and overlap, CHT, CHTs and CHTf have similar performances. All the trackers we proposed are however in the top 10 in the overall ranking in terms of accuracy and robustness (for both weighted mean and pooled mean). In terms of speed, our trackers are the fastest ones in EFO, outranking FoT [VM14]. In the category of real-time trackers, our method is only beaten by KCF [Hen+15], which is 6 times slower (in EFO). FoT [VM14] has similar performances in accuracy, robustness and expected overlap. If we compare our tracker with Hough-based ones, MatFlow is slightly more accurate and robust than our method, but is slower. Otherwise, all Hough-based methods are less effective and slower. Our tracker

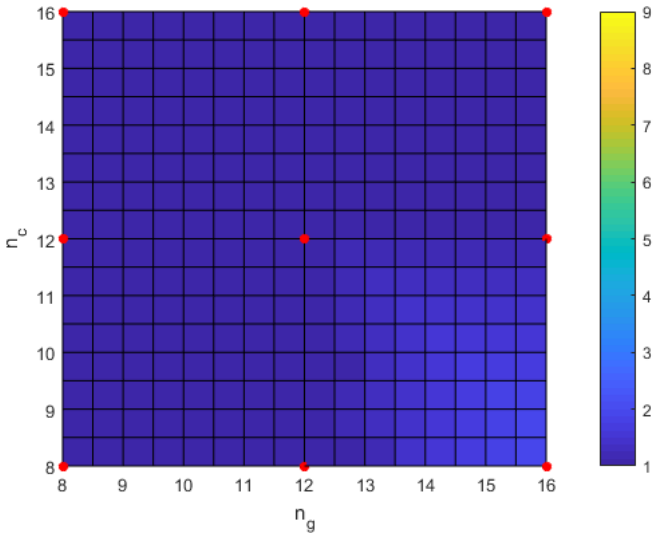


FIGURE 3.19: Accuracy rank for different values of n_c and n_g

main weakness is in terms of expected overlap, with 0.2944 for the best one (CHTf), which places it only in the first half of the VOT2014 competitors. Fig. 3.23 illustrates one correct tracking, while Fig. 3.22 shows a failure (due to fast motion of the target).

In terms of accuracy, our tracker behaves globally well: for each sequences, our tracker (in all forms) performs correctly and is correctly ranked (at least first half for all sequences). There are still some exception, such as *polarbear* (rank 32 among 40 trackers) and *ball* (rank 26), which may be caused by scale change. In terms of robustness, scale change issue is also a flaw of our tracker: for *fish2*, *hand1* and *hand2* sequences, our tracker performs badly (ranks 31, 31 and 29 respectively).

3.4.2.4 VOT15

VOT2015 [Kri+15] is composed of 60 sequences also taken from diverse datasets. Region noise experiments have been discarded, leaving only the baseline experiment.

Results are summarized Tab. 3.6. As for VOT2014, we use all results provided by the VOT committee (62 trackers), but we only show results for some trackers: MDNET [NH16], VOT2015's winner, real-time trackers [VNM13]; [MP14]; [VM14]; [MV11] and Hough-based trackers [HAS15]; [GRB13]. We also choose to show results of DSST [Dan+14a], previous winner, DAT [PMB15], Possegger's tracker based

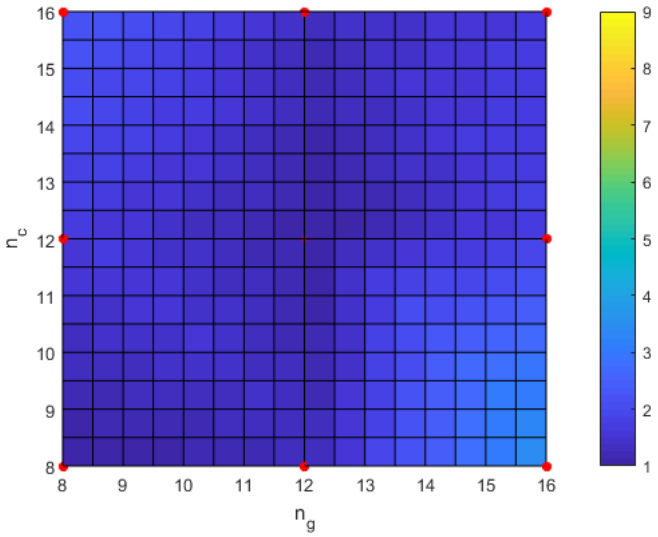


FIGURE 3.20: Robustness rank for different values of n_c and n_g

on color histogram exclusively, and STAPLE [Ber+16], for which results are available on author's webpage [3] (without speed results). Baseline corresponds to a virtual tracker obtained by averaging performances of state-of-the-art trackers published in ICCV, ECCV, CVPR, ICML and BMVC in 2014/2015.

As for VOT2014 [KPL+], our tracker is one of the fastest among all challengers (one of the few above 100 EFO). In terms of weighted ranking, it is very close to the best real-time tracker ASMS, Vojir's extension of Mean shift [VNM13]. We get 10% better in terms of expected overlap, but ASMS is better ranked in terms of accuracy and robustness. Otherwise, both are at the same level in terms of speed (EFO close to 110). We perform better than FoT which has similar expected overlap in VOT2014. Consequently, we can say that in the category of real-time tracker, ours is on par with the best competitors. If we look at the complete ranking, our tracker is still well ranked in weighted mean ranking (13th for the CHT and CHTs in both accuracy and robustness, top 20 and CHTf). In the pooled ranking, we are still in the first third in accuracy and robustness, except for CHTf in robustness (first half). Compared to similar trackers, Staple [Ber+16] is better ranked than our tracker: it may be due to the fact that its shape-based tracker is a more complex algorithm than ours (correlation filter based on HOG features), but is slower (according to the author, it reaches 80 fps on a 4.0 GHz on a desktop machine). Against

[3] http://www.robots.ox.ac.uk/~luca/staple.html

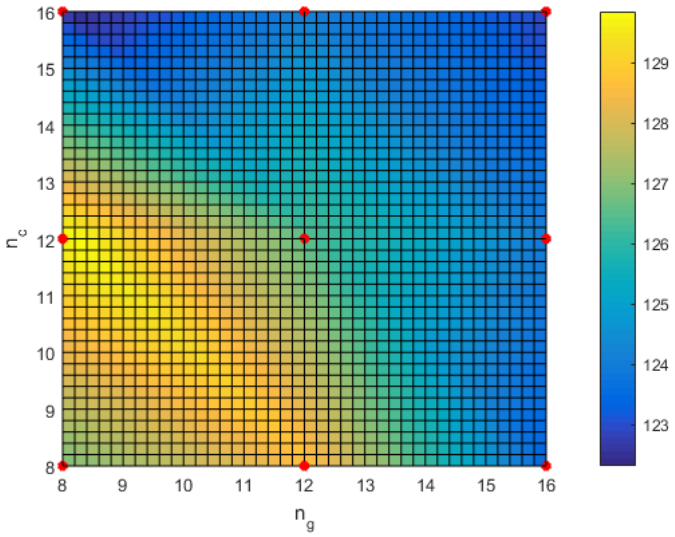


FIGURE 3.21: Speed for different values of n_c and n_g

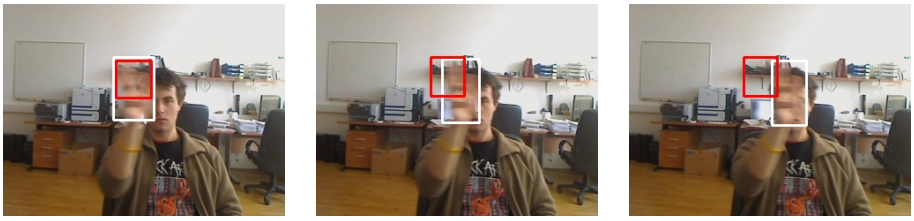


FIGURE 3.22: Frames from *hand2* sequence

DAT [PMB15], CHT and CHTs are better ranked in weighted mean and pooled rankings. However, substituting the distractor model of Possegger's by a GHT improves the expected overlap (8% higher for the lowest case). Otherwise, as for VOT2014, even though our tracker is well ranked in accuracy and robustness (see Fig. 3.25 for AR plot for only compared trackers), in terms of expected overlap, our tracker is only in the first half of the ranking.

Fig. 3.26 and Fig. 3.27 illustrate correct tracking even with high illumination change and object shape change. We also show Fig. 3.28 a case of correct tracking from the position tracker only, but due to no orientation estimation, obtained overlap is weak.

Regarding to per-sequence results, in terms of accuracy, our tracker (in all forms) is correctly ranked, and weak accuracy are, in most cases, due to scale change (as see for *tunnel*, *book* or *octopus*). Concerning robustness, scale change is still a weakness for our tracker. Otherwise, against sequences with high motion changes or deformations(*basketball*, *book*, *gymnastics3* or *iceskater*), our tracker performs very badly (below rank

Parameter	Notation	Value
Global parameters		
Surrounding area coefficient	α	2.0
Safe foreground area coefficient	β	0.20
Search window area coefficient	γ	$\sqrt{2}$
Color parameters		
Number of bins per channel	n_c	12
Updating rate	μ_c	0.06
Ratio for background area calculation	r_{bck}	2
GHT parameters		
Number of index of the R-Table	n_g	16
Updating rate	μ_g	0.08
Number of couples per entry	N_d	200
Gradient magnitude threshold	ϵ_g	70
State updating parameters		
Area variation threshold	ϵ_A	0.05
Max area variation	λ_A	0.05

TABLE 3.2: Parameters set

50 in robustness for all these sequences).

3.4.2.5 Analysis of results

Previously, we show results on academic datasets demonstrating that our tracker is at the level of the state-of-the-art. In this section, we will analyze more precisely the results, using the tools provided in the VOT toolkit.

One first remark concerns the performances of CHT, CHTs and CHTf. Surprisingly, the three of them have similar results for all the criteria. One reason may be the fact that parameters for scale and orientation are very restrictive, and do not tolerate high variations. Less tolerant values could create drift, leading to worse results. One solution may be to optimize all parameters related to scale and orientation estimations, to evaluate the relevance of the related methods. This remark leads us to test our tracker with different features using the position tracker only.

Another remark concerns the evaluation criteria. Given one sequence, trackers are evaluated in terms of speed, accuracy and robustness. The first criterion can be computed, and trackers can be ranked easily. But accuracy and robustness are correlated: let us consider performances in the sequence *sheep*, obtained by trackers with different methods to compute the gradient (Sobel, gradient at scales $\{1, 2, 4, 8\}$). Results are summarized Tab. 3.7. If we look at results in terms of accuracy, we obtain the

Tracker	Weighted mean rank		Pooled rank		Expected
	Acc.	Rob.	Acc.	Rob.	overlap
CHT	10.50	8.67	6	9	0.2963
CHTs	10.50	8.83	6	9	0.2891
CHTf	10.0	4.33	10	5	0.3014
bdf [MP14]	10.67	7.33	10	9	0.3148
DSST [Dan+14a]	1.83	4.33	1	3	0.3897
FoT [VM14]	7	17.83	6	23	0.3037
KCF [Hen+15]	2.0	4.67	1	5	0.3840
NCC	8.0	34	6	41	0.1574
Matflow [MP13]; [MP14]	8.67	3.17	6	4	0.3311
Matrioska [MP13]	9.83	12.33	6	9	0.2864
PTp [DG13]	26.17	11.17	32	9	0.2447

TABLE 3.3: Baseline results for VOT2014 (ranks over 43 candidates). CHT, CHTs and CHTf are our trackers.

highest results with CHT with a scale of 8. However, this good result can be due to the failure (the two trackers that fail are failing at the same frame) which leads to high accuracy afterwards (due to the re-initialization of the bounding box). Even with the fact that accuracy is not counted for 10 frames after a failure, the re-initilization can provide a certain advantage. Moreover, if the failure happened at different frames, or if different trackers fail a different number of times, comparing those trackers turns harder, as the re-initialization would greatly modify the accuracy. Therefore, we will only compare results of trackers when involved trackers are not failing at all, or are failing at the same frame.

Regarding to the biases of VOT datasets, the committee proposed the weighted mean rank to reinforce impact of rare difficulties. If we detail Illumination change and Camera Motion change, between the best and the worst ranked, number of failures are shown Tab. 3.8. We can see that in Illumination change, couples of failures can lead to high difference in accuracy (5 failure means a loss of 25 places) while in Camera Motion Change (very frequent in VOT15), it needs 45 failures to move from the first position to the last. This is more visible when we generate results from Tab. 3.6, as the most robust in terms of illumination never fails and the worst fails 17 times (for a ranking of 60 competitors), while in Camera Motion, the gap between the best and the worst is equal to 20 failures to 261. This suggests that a failure in illumination change has more impact in terms of ranking than one in camera motion.

Tracker	Weighted mean rank		Pooled rank		Expected
	Acc.	Rob.	Acc.	Rob.	overlap
CHT	8.33	5.17	8	7	0.2730
CHTs	8	5.33	8	6	0.2781
CHTf	9.17	4.00	8	6	0.2875
bdf [MP14]	7.83	7.33	8	7	0.3047
DSST [Dan+14a]	1.33	3.5	1	4	0.3490
FoT [VM14]	7.83	20.17	9	28	0.2681
KCF [Hen+15]	1.17	3.50	1	6	0.3443
NCC	11	37.33	8	41	0.1549
Matflow [MP13]; [MP14]	5.67	4.50	7	6	0.2930
Matrioska [MP13]	8.67	13.67	8	23	0.2479
PTp [DG13]	21.17	8.37	15	7	0.2592

TABLE 3.4: Region noise results for VOT2014

FIGURE 3.23: Frames from *torus* sequence

Using a code profiler, we managed to determine which functions are the most costly in our algorithm. The parameter that most impacts the computation time is the object size (which impacts the size of the search window). In most cases, the two most costly operations are the backprojection (Eq. 3.22), that can take 10% of the consumed time, and the GHT. For the backprojection, it is mainly due to the iterative pixel access. For the GHT, it is due to the gradient computation. This computation is made using OpenCV functions, based on floating point operations, more costly than integer functions. One remark we made previously, in Fig. 3.13, concerned the possibility to parallelize the computation of $\mathcal{GB}(\mathbf{HT}_t)$, $\mathbf{S}_{c,t}$ and $\mathbf{PR}_t$. As the GHT is more expensive than the two other operations

Tracker	Accuracy	Robustness	Expected overlap	Speed	
				EFO	fps
CHT	9.42	6.92	0.2846	115.16	141.29
CHTs	9.25	7.08	0.2836	146.83	180.15
CHTf	9.58	4.17	0.2944	144.82	177.69
bdf [MP14]	9.25	7.33	0.3097	46.82	100.45
DSST [Dan+14a]	1.58	3.92	0.3693	5.80	13.07
FoT [VM14]	7.42	19.00	0.2859	114.64	306.52
KCF [Hen+15]	1.58	4.08	0.3641	24.23	63.42
NCC	9.50	35.67	0.1561	3.95	7.80
Matflow [MP13]; [MP14]	7.17	3.83	0.3120	19.08	40.94
Matrioska [MP13]	9.25	13.00	0.2671	10.20	21.88
PTp [DG13]	23.67	9.92	0.2519	49.89	127.87

TABLE 3.5: Overall results for VOT2014

(at least 2 times), using three threads to compute them may not be the best solution. However, as computing $\mathbf{S}_{c,t}$ and $\mathbf{PR}_t$ takes approximately the same time, compute them sequentially in one thread, and computing the GHT in another thread may be a better solution.

The first experiment we made concerns relative performances of the two parts of the tracker (Hough-part and color-part), by testing each part of the tracker on VOT2015. Expected overlap shown Tab. 3.9 already indicates that isolated, Color and Hough do not perform well. Fig 3.29 shows results in terms of accuracy and number of failures according to different difficulties. Results are close to what we expected, as the color tracker only is the tracker with the weakest overlap score in case of illumination changes (and is also close to the Hough tracker only in terms of failure). Globally, the GHT alone does not produce a robust tracker (for all difficulties).

We also studied the impact of different color spaces. We chose the RGB color space as a reference (still with the position tracker only), and compared it with the three following color spaces:

- No color. Since usually, R, G and B channels are correlated on cameras, it is relevant to test performances of our tracker using only the grayscale channel

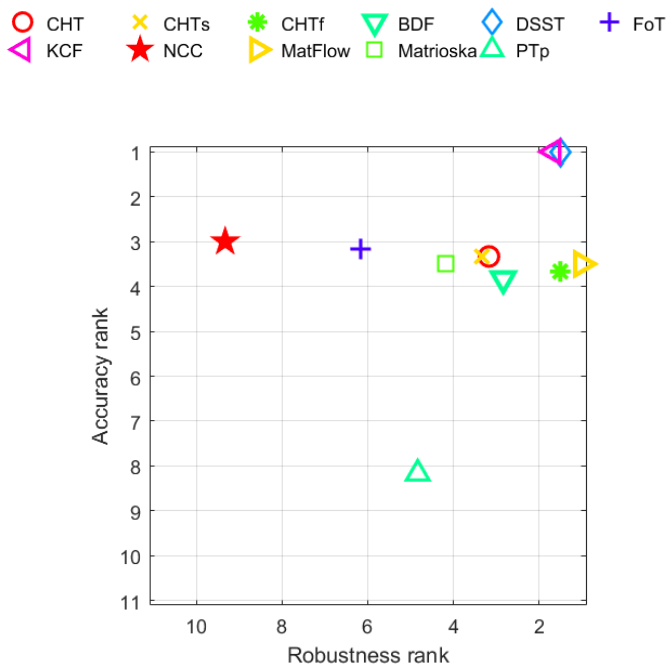


FIGURE 3.24: AR plot for trackers displayed on Tab. 3.5

- HSV color space, for which we only work on Hue-histogram. In this case, histograms are computed by discarding weak values of saturation (below 10). This tracker will be denoted HSV

- Lab color space, for which we work on ab-histogram (denoted ab) and lab-histograms (lab). For ab case, we will discard pixels with Luminance below 50 in histogram computation. We tested the ab color space to see how important the luminance channel was, and to see relevance of opponent color theory in object tracking

- Color attributes proposed by [VDW+09], already mentioned in Section. 2.1.1 and used for Content-based Image Retrieval. In the code proposed by Van de Weijer [4], RGB pixels are quantized into a $32 \times 32 \times 32$ space. Then, a function from $\{0, 1, \ldots, 31\}^3$ to $\{0, 1, \ldots, n\}$ is created (Van de Weijer proposed two functions, one with 11 colors, another one with 50). The start space is the quantified RGB color space, and the end space represents attribute colors. The built histogram then represents the distribution of color attributes. We performed the test with $\mu_c = 0.06$ (as for other color spaces) and with $\mu_c = 0.02$. Trackers will be denoted "Att*numcolor* (μ_c)" (Att50

[4] http://cat.uab.es/~joost/software.html

Tracker	Weighted mean rank		Pooled rank		Exp. overlap	Speed	
	Acc.	Rob.	Acc.	Rob.		EFO	fps
CHT	12.83	13.50	8	20	0.2668	115.90	124.07
CHTs	13.00	13.17	10	17	0.2664	118.25	126.59
CHTf	12.00	19.17	5	28	0.2613	114.82	122.93
Staple [Ber+16]	1.00	4.33	1	5	0.3450		
ASMS [VNM13]	7.50	10.50	2	13	0.2353	115.09	142.26
Baseline	4.67	16.00	2	20	0.2353	1.01	2.32
bdf [MP14]	29.50	31.83	27	44	0.2054	200.24	78.43
DAT [PMB15]	13.67	17.33	6	20	0.2428	9.82	14.87
DSST [Dan+14a]	4.00	23.67	1	36	0.2707	3.29	4.47
FoT [VM14]	19.67	45.67	16	53	0.1934	143.62	177.53
HT [GRB13]	20.00	28.00	13	44	0.2045	0.91	0.56
Matflow [MP13]; [MP14]	22.17	27.50	23	44	0.2098	81.34	31.86
MDNET [NH16]	1.00	1.33	1	1	0.3789	0.87	0.97
NCC [Lew95]	8.33	64.83	2	66	0.1359	172.85	105.25
sPST [HAS15]	1.67	4.50	1	5	0.3134	1.03	1.16

TABLE 3.6: Overall results for VOT2015 (ranks go to 66)

($\mu_c = 0.02$) will represent the color attribute with 50 colors and $\mu_c = 0.02$)

We are using the same parameter set as Tab. 3.2: grayscale and Hue histogram are composed of 12 bins, and the ab-histogram is sized to 12×12 bins. First, in terms of accuracy, as for parameters optimization, all trackers have similar performances. The difference is made in terms of robustness (see Tab. 3.10 for details). Second, we can note the weakness of the tracker ab. Indeed, it performs very badly in terms of overlap and accuracy for most sequences and more particularly on *glove*, *ball2* or *car1* (results were obtained using *report_article* function from the VOT-toolkit,

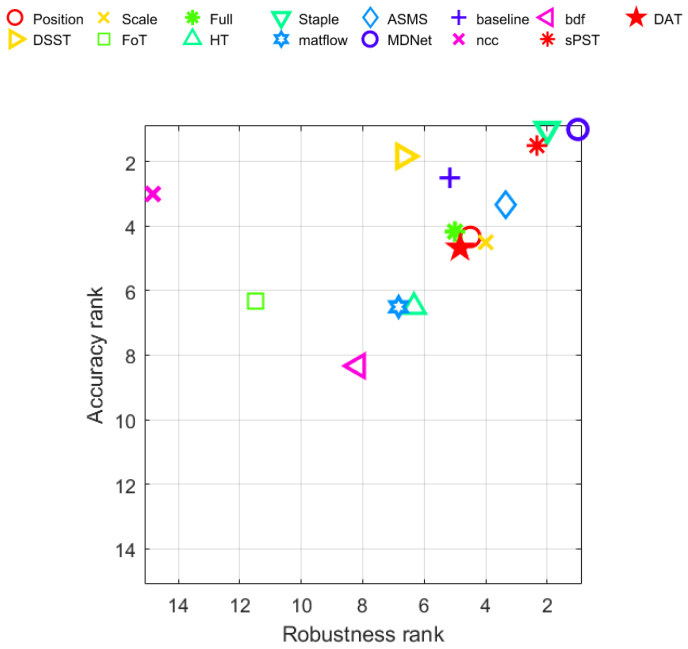


FIGURE 3.25: AR plot for trackers displayed on Tab. 3.6 for VOT15

FIGURE 3.26: Frames from *matrix* sequence

providing accuracy and robustness for each sequence, and available in Appendix. A). In Tab. 3.10, we can see that in terms of robustness, it performs very badly compared to other trackers, excepted the two Att50. This is true for all difficulties. It is mainly because for these particular sequences, the ab color space does not provide a sufficient power of discrimination to separate object pixels from the background: for *ball2*, the problem is that black and white are not different from blue in ab space, resulting in non observable object. For *car1* and *glove*, the issue is related to the blueish aspect of the target and the surrounding background. On the contrary, ab is performing very well, and even better than all other algorithms, for the *helicopter* sequence. This is due to obvious separability of the helicopter (red) and background (green grass and blue sky) colors. Fig. 3.30 illustrates these cases.

FIGURE 3.27: Tracking from *butterfly* sequence

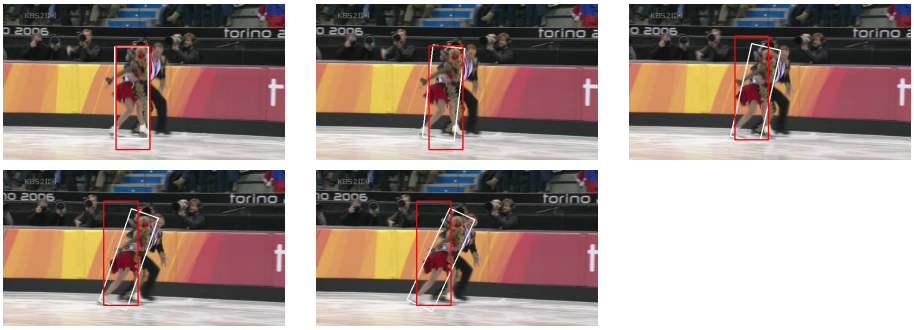


FIGURE 3.28: Tracking from *iceskater2* sequence

If we add the Luminance channel, results are getting much better, and closer to those from RGB channel. However, due to the low score of ab (highlighting the low power of discrimination of the ab color space), we can expect that the lab color space leads to slightly worse results than RGB. For HSV based tracker, performances are slightly lower than the reference (CHT). The size of the histogram (12 bins against 12^3) may be the cause of the slight drop of accuracy, but, the color model is 12^2 lighter (in terms of memory consumption) than for the original tracker. The last remarks concern Van de Weijer color attributes [VDW+09]. In all cases (number of color, value of μ_c), performances are low. Weaknesses are due to two reasons:

- The instability of color measure over the time. From one frame to another, pixel color attributes can drastically change, while the variation is smoother in traditional color spaces. Fig. 3.31 illustrates fast variation of color attributes of the target (the plastic bag) whose pixels are moving from the attribute 'white' to the attribute 'blue'. It is noticeable for Att50, when we compare $\mu_c = 0.06$ which gives

	Overlap	Failure
CHTSobel	0.49	0
CHT	0.53	0
CHTscale2	0.52	0
CHTscale4	0.47	1
CHTscale8	0.55	1

TABLE 3.7: Overlap and number of failure for different trackers in the sequence *sheep*

Rank	Cam	Ill
1	50 - 54	1 - 4
Last	98 (rank 36)	9 (rank 26)

TABLE 3.8: Number of failures per rank on Camera Motion an Illumination change

an overlap of 0.1999, while $\mu_c = 0.02$ (meaning weaker update of the color histogram) gives 0.2116. In the code provided by Van de Weijer, two implementations are available: one mapping directly a RGB bin to a color label, and one giving the probability that this bin leads to each attribute. This second implementation may be tested in the future

- The lack of color to describe the target. Fig. 3.32 illustrates one case where the target color (the left cat) is similar to the background. This remark holds for HSV space too. However, in this case, as the Hue histogram is more stable than color attribute histogram, the loss of performance is less dramatic. Interestingly, Att50 performs very badly in terms of robustness. This remark should be a topic for further discussions. In terms of difficulties, its main weaknesses is Camera motion

The first series of tests concerned the impact of visual color features. Surprisingly, the RGB color space gives the best results. Other ones suffer from different weaknesses: the Hue space in HSV suffers from the lack of dynamic of the color space (even though impact of this flaw is limited, with respect to the final expected overlap), Lab suffers from the weak power of discrimination of ab color space, and color attributes are unstable over the time. We can now move to the second series of tests.

These tests concern the geometry features. CHT still serves as a reference, and can be identified to a Gaussian derivative computed at scale 1.

Tracker	Expected Overlap
CHT	0.2668
CHTs	0.2664
Full	0.2613
Color	0.2126
Hough	0.1854

TABLE 3.9: Expected overlap for different forms of [TM17]

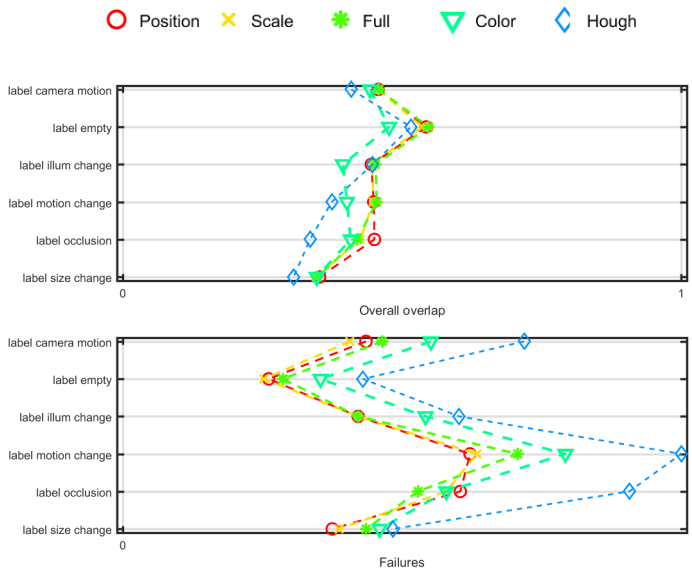


FIGURE 3.29: Accuracy and failure for different versions of CHT

We compare it to gradient computed with Sobel filter (with kernel of size 3×3) and gradient computed with Gaussian derivative formulation, with σ in $\{2, 4, 8\}$ (Fig. 3.33 illustrates gradient magnitude at those scales), and with kernel sizes equal to $2 \cdot \sigma^2 + 1$. Complete results per sequences are available in Appendix. B. However, in our case, the goal is to see impact of different scales for derivative computation.

Again, all trackers have the same accuracy rank (except Sobel, which is ranked 1.83), and differences are in robustness. The largest scaled tracker give relatively weak results compared to other scales, but still gives satisfactory results (given the expected overlap, it performs as well as the extension of the Mean Shift [VNM13], and as well as DAT [PMB15]). The bad rank obtained in terms of robustness to Illumination change (12 failures, while others fails at max 7 times) induced the bad rank in robustness in the weighted mean. If we study particular sequences, one clear weakness is also the size of the target: objects smaller than gradient

Tracker	Weighted mean rank		Pooled rank		Expected overlap
	Acc.	Rob.	Acc.	Rob.	
CHT	1.0	1.17	1.0	1.0	0.2668
ab	2.0	7.33	1.0	9.0	0.1752
Grayscale	1.0	1.83	1.0	9.0	0.2166
HSV	1.0	1.33	1.0	3.00	0.2281
lab	1.0	1.67	1.0	2.0	0.2519
Att11 ($\mu_c = 0.06$)	1.0	2.33	1.0	3.0	0.2085
Att50 ($\mu_c = 0.06$)	1.0	5.33	1.0	7.00	0.1999
Att11 ($\mu_c = 0.02$)	1.0	2.67	1.0	3.0	0.2069
Att50 ($\mu_c = 0.02$)	1.17	5.33	1.0	7.0	0.2116

TABLE 3.10: Results for color variant trackers

	Expected overlap
Scale 4	**0.2732**
Scale 2	**0.2713**
CHT	0.2668
Scale 8	0.2389
Sobel	0.2262

TABLE 3.11: Expected overlap for geometrical variant trackers

	Weighted mean		**Pooled**		**Speed**	
Tracker	**A-Rank**	**R-Rank**	**A-Rank**	**R-Rank**	EFO	fps
Position	**1.00**	**1.17**	**1.00**	**1.00**	115.90	124.07
Sobel	**1.83**	**1.33**	**1.00**	**2.00**	90.54	96.83
Scale 2	**1.00**	1.50	**1.00**	**1.00**	86.38	92.48
Scale 4	**1.00**	**1.17**	**1.00**	**1.00**	61.94	66.31
Scale 8	**1.00**	2.50	**1.00**	4.00	52.28	55.97

TABLE 3.12: Accuracy for geometrical variant trackers

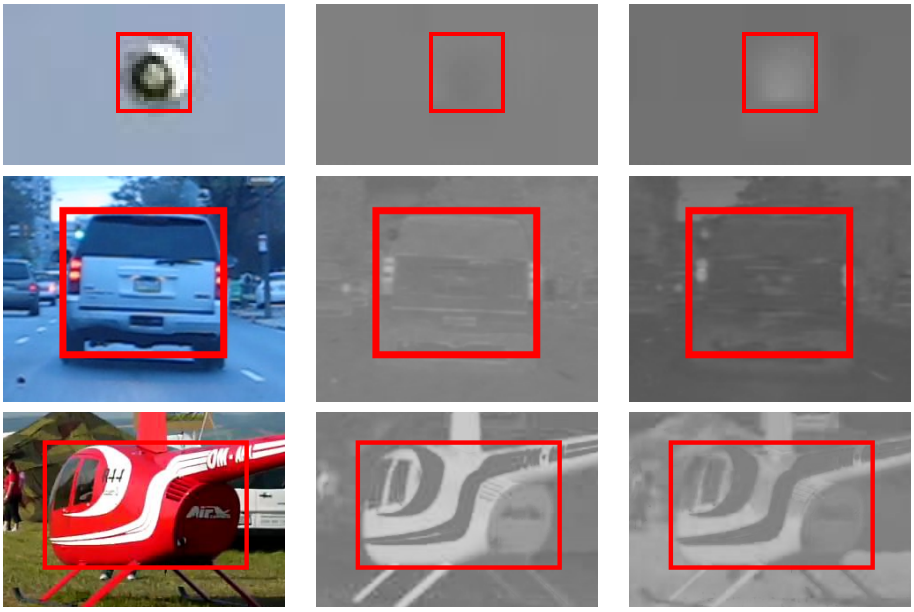


FIGURE 3.30: Frames from *ball2*, *car1* and *helicopter* and their projection into channel a and b.

scale are not correctly represented. We can for example consider *ball2*, in which Scale 8 leads to an average overlap of 0.02, while others manage to reach more than 0.70, or *handball1*, for which it fails 6 times, twice more than in other cases. Another question concerns the possibility to use a characteristic size to track a target. The idea would be to smooth all non relevant details to keep only the sufficient information to track the target. The existence of such best scale (and of a method to estimate it), may improve tracker's performances (neglecting the object size change). So, if σ_{opt} is the characteristic scale to compute the gradient, then σ_{opt} should be a maximum for at least the overlap criterion. However, if we consider the sequence *road*, where the target keeps its initial size (difficulties of this sequence are mainly occlusion and camera motion), the most accurate tracker is obtained with the largest scale (8), even though it is the only one to fail (2 failures) (Tab. 3.13 shows overlap results, and shows that the overlap in function of the scale does not have the wanted monotony, as $\sigma = 2$ is a local maxima). This is all the more surprising that the target is small (width of 30 pixels), and we can expect that low scales are more adapted. Intermediate scales (2 and 4) are at the same order of magnitude in terms of robustness. In terms of expected overlap, they are more effective than the tracker with scale 1. In the whole VOT2015 ranking (with results provided by the committee and by Staple's author [Ber+16]), the tracker with a scale of 4 would be ranked 23

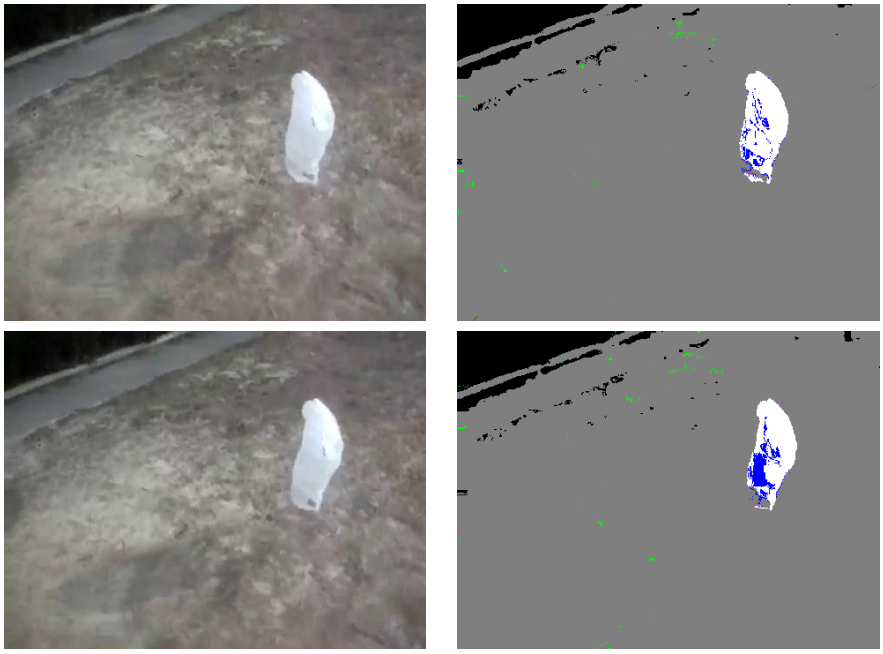


FIGURE 3.31: Frames 9 and 10 from *bag*, with their color attributes mapping.

FIGURE 3.32: Color attribute from *fernando*.

in terms of expected overlap (among 63 competitors), two positions above DSST (VOT2014 winner). In terms of speed, larger scales are slower than lower ones (computation cost of derivatives made by Gaussian kernels of larger sizes, using OpenCV implementation). However, in all cases, speed is still satisfactory (more than 50 fps). Rather than searching for an optimal scale per sequence, one solution to improve the accuracy, and that we can test at short-term, should be to use a multi-scale GHT.

Finally, the last remark concerns the use of Sobel filters, whose main interest was the computation time. However, considering the results (in terms of speed, accuracy and robustness), replacing the Sobel filter by a Gaussian derivative for tracking is a right decision.

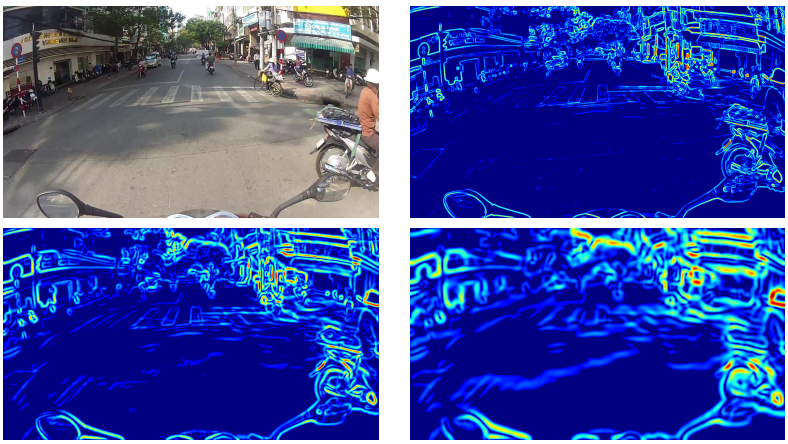


FIGURE 3.33: Gradient magnitude for $\sigma \in \{2, 4, 8\}$

	Overlap
Position	0.61
$\sigma = 2$	0.65
$\sigma = 4$	0.60
$\sigma = 8$	0.68

TABLE 3.13: Overlap of road for different values of σ

3.5 Conclusion

We presented our work in object tracking. From the preliminary works [TM15], which combines GHT and Particle Filter, to the final tracker [TM17], which is still based on GHT, but for which the Particle Filter was replaced by a simple color histogram used to build a pixel confidence map, we managed to propose algorithms based on pixel color and gradient exclusively. In both cases, the GHT is applied in its original form [Bal81], unlike trackers from the literature which rely on more complex GHT (based on more complex features or codebooks) The last proposed tracker was tested on academic datasets VOT14 and VOT15 [KPL+]; [Kri+15], and gives decent results in terms of accuracy and robustness (top 10 in accuracy and robustness for VOT14, top 20 for VOT15). In terms of speed, our non-fully optimized tracker can run at more than 100 fps on a laptop machine at 2.4 GHz, without explicit multi-threading. This tracker is then one of the fastest from the state-of-the-art, and is still competitive on modern datasets. Compared to other Hough-based trackers, only Hua [HAS15] proposed a better tracker, but with a much higher computation time. Otherwise, Matflow [MP13]; [MP14] is a better ranked Hough-based tracker in VOT2014. However, [TM17] performs

better in VOT15. Pixeltrack [DG13] is the fastest Hough-based tracker on VOT2014, and is very similar to our tracker. However, we managed to outperform their results in terms of accuracy, robustness, overlap and speed.

We achieved our goal to work on light and low-level features (pixel colors and derivatives) and simple implementation of the GHT. Results are very satisfactory on academic datasets. We also optimized a subset of the parameter set. As we succeed to separate the color-based tracker from the shape-based one, our tracker also served as a base to determine impact of different features. We tested impact of changing color space (HSV, lab, color attributes [VDW+09]) and different scales of gradient computation. From these different tests, we concluded that the RGB color space, in spite of limitations mentioned in Chapter. 2, presented the best performances in terms of accuracy, robustness and overlap. In terms of gradient computation, high scales produce relatively weak results. The best results are obtained with intermediate scales ($\sigma = 2$ or $\sigma = 4$).

Now, in terms of perspectives, we have highlighted some limitations and propose possible enhancements of our work:

- As mentioned before, the scale and orientation estimation is not satisfactory: results are not better than position tracker only. With the current method, the solution should be to optimize all the parameter related to scale and orientation. Another solution would be to compute more complex operations, such as segmentation, or by setting dynamically the value of $\epsilon_{\mathbf{BF}_t}$ (threshold value used to determine which pixels in $\mathbf{BF}_t$ belong to the target). If the confidence map $\mathbf{BF}_t$ is reliable, the first choice can be a good solution as, in the literature, several segmentation algorithms can do the task effectively, at the cost of an increased computation time. The second choice may involve a function to estimate tracking confidence to adjust $\epsilon_{\mathbf{BF}_t}$ according to the confidence. It should be faster to compute, but less reliable than a segmentation

- The parameter tuning is not complete: we optimized only 4 parameters that we considered critical. As mentioned earlier, parameters related to scale and orientation estimation can also be optimized in our full tracker

- In terms of feature space, our tracker can perform differently, according to the scale and the color space used. One possible way

for improvement is selecting the "correct" color space, knowing the sequence. We can for example cite Collins [CLL05] who selects the color space optimizing a measure of discrimination based on likelihood. By considering an optimal tracker, which has, for each sequence, the results from the tracker with the right color space, we can see how good Collins' method can predict this optimal color space. Concerning scale, we already mentioned the idea to define the optimal scale, knowing the sequence. We can also think about using the Hough Transform at multiple scales for derivative computation

- Our algorithm is light and fast. From a practical point of view, it can serve as a base for many enhancements: detection of failure (and recovery), model updating with μ_c and μ_g varying over the time. While the first enhancement can be done by combining the tracker with a detector (in that case, we are close to object specific and multi-target tracking), the second one can imply a measure of confidence, which can be related to the first point (scale and orientation estimation) In that sense, we can expect to improve our tracker by diverse extensions

Even though the VOT committee provides us different methods to compare and to rank different trackers, the task is still complicated as their two main criteria (accuracy and robustness) are correlated, and comparing the raw performances in accuracy can not be done independently to robustness. To have a better view of performances, we can still test our tracker on different datasets: VOT2016 [Kri+16b], OTB [WLY13], Temple-Color [LBL15]... We can also test [TM17] on the VOT-TIR dataset, with sequences taken with thermal infrared sensors. All of them propose different sequences (or in VOT2016 case, different annotations), different methods of evaluations, and can lead to further analyses of our tracker. If we want to remain on actual tested datasets, results were obtained with parameters tuned according to the weighted mean rank. As the VOT committee also proposes another ranking, the pooled one, making the same study with this other ranking may be interesting. Pooled ranking also has another interest: as trackers are ranked according to sequences and not by difficulties, we can study performances sequence per sequence, and no longer according to difficulties.

To conclude this part, combining the original GHT with a foreground background discrimination model based on RGB-histogram leads to a very fast tracker, with decent accuracy and robustness. It means that, for tracking, this level of representation is sufficient to discriminate objects from background. From a systemic point of view, this tracker is very light, fast and accurate, then suitable for low-power systems. In the next chapter, we focus on object detection. Feature spaces are designed in order to describe classes of object in the most generic way possible. The interest will then be to see how far we can go, using basically the same representation as we used for tracking.

Chapter 4

Object Detection

Contents

4.1 Literature review 117
4.1.1 Object classification, detection, recognition . . 117
4.1.2 Hough detectors 119
4.2 Hough Forest 120
4.2.1 Forest training 120
4.2.2 Extensions of the Hough Forest 126
4.3 Our contributions 128
4.3.1 Patch generation 128
4.3.2 Node training 129
4.4 Experiments 132
4.4.1 Evaluation method 132
4.4.2 UIUC Cars 134
4.4.3 TUD Pedestrian 135
4.5 Conclusion 140

Introduction

The second task we will present is object detection. It consists in defining a class of object (human face, car, pedestrian), and localize instances of this class in test images. Object detection can be used in many areas:

- For statistics application. For example, counting cars moving on roads, counting passengers going through a door or a corridor
- For surveillance: detecting people entering secured areas

- In biometrics, a face recognition system may require a step of face detection, in order to analyze it. We can mention iris as a biometric modality, which also requires eyes detection
- In photography: detecting faces to adjust the focus of the camera, or detecting smiles

Object tracking (presented in the Chapter. 3) and object detection are very different tasks. Indeed, in terms of features, those used for detection have to describe the class of object as precisely as possible, coping with intra-class variation. Fig. 4.1 illustrates different instances of the class "pedestrian": shape, color, size aspect are all different from one instance to another. For tracking, the feature space can be chosen in order to discriminate target from background (such as [TM17]), and to be resilient to object deformation or context change.

To cope with intra-class variation, detectors have to learn visual aspects of many objects. Machine Learning methods are then adapted for this task.

FIGURE 4.1: Instances from pedestrian class

In the framework of our thesis, exploring limits of low-level features in object detection is interesting. In accordance with the central role of the Hough Transform in our thesis, we will base our work on the Hough Forest proposed by Gall [GL13]. Our positioning with regard to Hough-based detectors will be presented in this chapter, which is organized that way:

- First, we will make a short review of literature
- Second, we will present Hough Forest algorithm [GL13], some implementation details provided by Gall, and then a review of Hough Forest-based detectors

- Third, we will present our works [TM16]. As we planned to reduce the dimension of the feature space, we need to compensate with additional information.
- Fourth, we will present results on academic datasets

4.1 Literature review

In this section, we present popular works on object detection. In the first part, we present different algorithms. Then, we will focus on object detectors based on Hough Transform.

Formally, object detection is close to a 2-class classification task, which consists in, given an image, determine if it belongs to the learned class (foreground) or not (background). One difference is that detection also implies a localization task meaning that, the aim is also to estimate instance's state (position and scale for example). Moving from classification to detection can be done with a sliding-window approach, by dividing the image into several sub-images (of different sizes, at different positions) and compute a classifier on them.

Object detection is also one special case of object recognition, which combines localization and image classification (multi-class classification task). Consequently, we will enlarge our review to object recognition and to object classification.

4.1.1 Object classification, detection, recognition

Object detectors rely on machine learning classifiers. There are many types of classifier, and we will not propose a full coverage of the problem. We will rather present some popular classifiers, and then finish with state-of-the-art works.

Viola and Jones proposed a very popular face detector [VJ01]. Their contributions rely on three important elements. First, their feature space is generated using Haar wavelets, and is quickly obtained using integral images. Second, by using AdaBoost [FS95], they obtain a classifier relying on a small number of dimensions in terms of feature space. Third, the cascade architecture of their classifier, with weak and fast classifiers at the top and strong and slow at the bottom, allows to quickly discard a large proportion of false alarms. This contribution is important, since Viola and Jones were among the first to propose a real-time detector.

Another popular classifier, originally designed for pedestrian detection was proposed by Dalal and Triggs [DT05]. The important part of the contribution is the HOG features, already presented Section. 2.1.2.4, and present in many algorithms (including Hough Forest [Gal+11]). This contribution is all the more important that, by using a simple SVM for detection, Dalal demonstrates that designing high-level regional features provided good results for detection.

In the category of tree decision-based detectors, Moosmann [MNJ08] proposed an algorithm for object classification, structurally closed to Hough Forest. Moosmann uses random forests, also composed of binary trees. However, his works only deals with image classification. We will detail similarities and differences in Section. 4.2.

In the category of state-of-the-art detectors, Convolutional Neural Networks are now the reference in terms of accuracy. Deep Learning became popular in object classification, when researchers, notably Krizhevsky [KSH12] managed to beat the state-of-the-art on classification in ImageNet dataset [Den+09] using CNNs to train the network AlexNet, outranking previous methods based on Bag of Words [Csu+04], and reaching performances in image classification close to the human [Rus+14]. AlexNet was followed by different CNNs such as Overfeat [Ser+13], VGG [SZ14]... Girshick [Gir+14] managed to beat state-of-the-art detectors on Pascal VOC datasets [Eve+07]; [Eve+10] by proposing R-CNN. From an image, Girshick extracts many search windows. Each candidate window is then tested using CNNs to determine if the image belongs to one class (Pascal VOC contains several classes). Girshick also considers two CNNs: one for classification, using ImageNet for training [Den+09], one for localization (using a subset of Pascal VOC for training). Many works based on R-CNN aimed to enhance training and detection speed [Ren+15]; [Red+16], or to enhance accuracy [ZD14] (for this category, [HBS14] proposed a survey of different contributions).

Before Deep Learning methods, Deformable Part Model (DPM) [Fel+10] was the reference in object detection. We already mentioned this paper in Chapter. 3 as Danelljan [Dan+14a] used PCA-HOG as a feature space (combination of HOG features, computed with 9 orientations and 4 normalization factors, and PCA to reduce the dimension of descriptors). DPM relies on the filtering operation, computed at different scales. One object is described by a main filter, and by different parts, each part itself

modeled by its own filter, by an offset vector (defined by the relative position of this part with regard to object's center), and by a vector defining a deformation cost. Detecting an object is then realized by searching the area having high response on all defined filters. For object detection task, a latent SVM is then trained.

4.1.2 Hough detectors

In this section, we focus on detectors using Hough Transform. Compared to the original GHT [Bal81], the main principle consists in replacing the R-Table by a codebook. The limit of the R-Table is that all lists of displacements are only indexed by gradient orientation. This limit has already been considered in tracking case, as Godec [GRB13] replaced it by random Ferns. In detection task, the problem is more critical as we do not consider single object in detection, but classes of object. In that case, higher level features, with higher power of representation, have been designed for detection: HOG [DT05], keypoints [Low99].

Leibe's Implicit Shape Model (ISM) [LLS08] is one of the first object detector exploiting the GHT, and extending the notion of R-Table to a codebook indexed by higher-level features than gradient orientation. Leibe represents each image from the training set with a set of keypoints (by testing different types of keypoints, and among them, SIFT [Low99]). Each keypoint is also associated to a displacement from the keypoint to a reference point. Then, Leibe regroups all similar features using clustering method. By clustering the keypoints set, Leibe generates a codebook, taking the role of the R-Table in the original GHT. In detection mode, given an image, keypoints are extracted, passed through the trained codebook, and vote according to the resulting entry. Then, after localizing object (by maxima search), Leibe also proceeds to a backprojection of maxima, to accurately segment detected object.

The Max-Margin Hough Transform [MM09] proposed by Maji extends the training step by applying a step of weight training, to maximize weight of positive images. For the detection step, after its Max-Margin Hough Transform, Maji also uses another classifier to validate the detection.

By grouping images from training set in different latent classes, Razavi [Raz+12] proposed the Latent Hough Transform. The problem tackled by Razavi is then to solve an optimization problem in order to get the

optimal latent matrix, representing the method to group images from the training set. This approach has also been used by Tejani [Tej+14], who combines Hough Forest [Gal+11] and latent variables formulation to 3D object detection.

In Chapter 3, we mentioned the backprojection operation. In object detection case, this operation is very helpful, as shown by Razavi [RGVG10]. The key is to consider one detected instance (obtained by local maxima search), and to backproject this detection to get the support of votes (i.e. the set of elements which have voted for this instance). Compared to the backprojection mentioned in the tracking section, used by Duffner [DG13] and Godec [GRB13], Razavi's backprojection is softer, as he backprojects the peak and its surrounding too: leading to a non-binary backprojection map (compared to Eq. 3.8). This backprojection can be used in many ways:

- To estimate the bounding box
- To measure the similarity between detected instance and training objects
- To measure the similarity between two detected instances

Another important variation of the Hough Transform in object detection context is the Hough Forest, proposed by Gall [GL13]. Due to its importance for our work, we will make a full presentation of this algorithm in the next section.

4.2 Hough Forest

In this section, we present Hough Forest, as proposed by Gall [GL13]. This algorithm, based on Random Forest [Ho95]; [Bre01], is very close to one object recognition algorithm proposed by Moosmann [MNJ08]. Our explanation will be divided in two parts:

1. Forest training
2. Object detection

4.2.1 Forest training

In this part, we explain the forest training. Our explanation is made in three steps: first, how to generate training set, given input images. Second,

how to train forests. Third, some implementation details provided by Gall [GL13] are presented. All explanations are done for the training of one unique tree. The training of the whole forest basically consists in repeating the process N times.

4.2.1.1 Generating training set

Training a detector (and more generally, a classifier) requires the generation of training set. In detection case, given one class of object (car, human, animal...), the principle consists in considering positive images (image from this class) and sometimes negative ones, and to project them into a feature space. For example, Viola and Jones' face detector [VJ01] relies on Haar wavelet features to produce a very fast face detector. Leibe trains its classifier using a sparse representation, based on keypoint features [LLS08].

Hough Forest algorithm is similar to Moosmann's method [MNJ08]: both consider positive and negative images, and from each image of these two classes, extract patches, randomly chosen. All patches have the same size $w \times h$, and are projected into a multi-dimensional feature space:

- The 3 channels from Lab color spaces
- Absolute value of first derivatives $|\frac{\partial \bullet}{\partial x}|$ and $|\frac{\partial \bullet}{\partial y}|$
- Absolute values of two second order derivatives : $|\frac{\partial^2 \bullet}{\partial^2 x}|$ and $|\frac{\partial^2 \bullet}{\partial^2 y}|$
- HOG features [DT05] generating 9 components: one for each bin

those computed features results to 16 dimensions feature space. Morphological erosion and dilation are then computed on each feature, leading to a 32D feature space. The aim of these operations is to add robustness toward noise and small deformations. Moosmann, in his case, tested different feature spaces: based on color (HSL color space), Haar wavelet or grayscale SIFT descriptors.

We will denote, for each $(x, y) \in \{0, 1, \ldots, w-1\} \times \{0, 1, \ldots, h-1\}$ and $c \in \{0, N_d - 1\}$, by $\pi_c(x, y)$ the value of the patch π, at coordinate (x, y) and for the feature c. Positive patches are additionally described by their relative position with respect to object's center. Fig. 4.2 illustrates patch sampling from two images of UIUC Cars training set [AAR04]. Blue patches are negative, red ones positive. For this whole chapter, we will denote as $\mathcal{P}^+$ and $\mathcal{P}^-$ the positive and negative patch sets used to train the detector. Let us also define $\mathcal{P} = \mathcal{P}^+ \cup \mathcal{P}^-$.

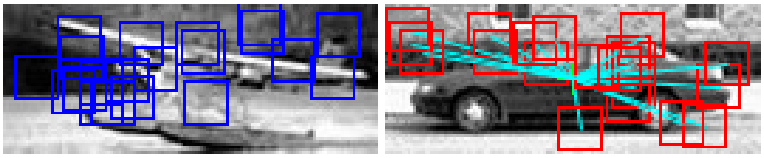


FIGURE 4.2: Patch sampling from images from UIUC dataset [AAR04].

4.2.1.2 Training tree

Now that the training set for one tree has been generated, we now move to the training of the Hough tree. Let us first provide useful notations. Each Hough Tree T is a binary tree. Let d_T be its maximal depth. T is then composed by at most $N_T = 2^{d_T} - 1$ nodes. These nodes are divided into two types, the leaves (terminal nodes) and the non-leaves. All nodes are stored in a table, and each node is indexed by $i \in \{0, 1, \ldots, N_T - 1\}$. We can then code a tree as a list $T = \{T(0), T(1), \ldots, T(N_T - 1)\}$. Given a non-leaf node of index j, its two child nodes are indexed by $(2 \cdot j + k)$, with $k = 1$ for the left child node, and $k = 2$ for the right one. Fig. 4.3 illustrates an example tree of depth 3. Green circles symbolize nodes, blue squares leaves.

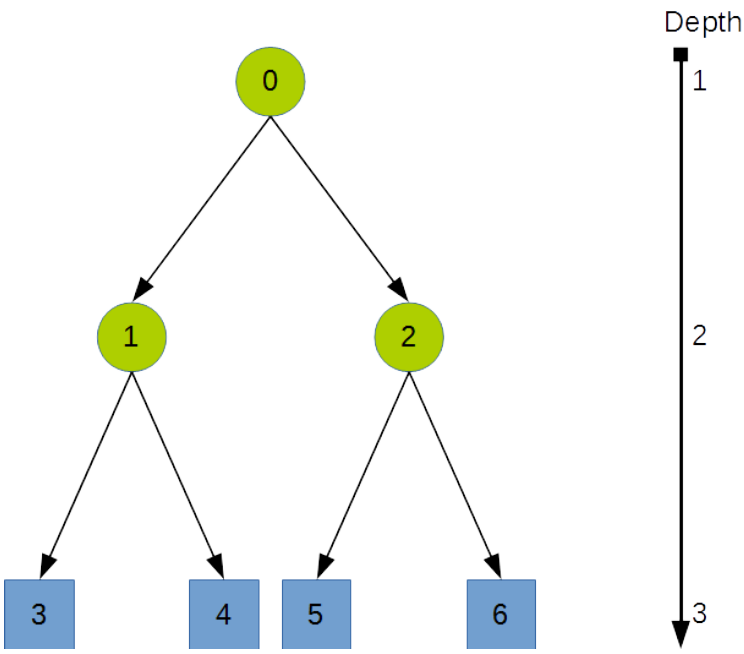


FIGURE 4.3: A binary tree of depth 3.

For a training set $\mathcal{P}$, each non-leaf node j is trained by considering a subset $\mathcal{P}(j)$ of $\mathcal{P}$ in order to split it into two disjoint sets $\mathcal{P}(2 \cdot j + 1)$ and $\mathcal{P}(2 \cdot j + 2)$, used to train the two nodes $T(2 \cdot j + 1)$ and $T(2 \cdot j + 2)$. We

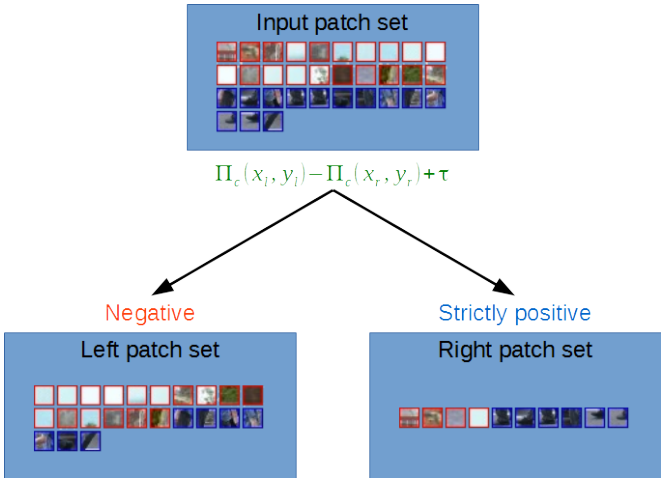


FIGURE 4.4: Patch set split into two disjoint parts

also have $\mathcal{P}(0) = \mathcal{P}$. The split process is done by a binary test, designed according to an optimization procedure that will be detailed further. This binary test is a function taking as an input a patch $\pi \in \mathcal{P}(j)$ and using 6 parameters:

- two couples of coordinates $p_l = (x_l, y_l)$ and $p_r = (x_r, y_r)$
- One channel c
- One threshold value τ

and defined by the quantity:

$$\mathcal{Q}(\pi) = \pi_c(x_l, y_l) - \pi_c(x_r, y_r) + \tau \tag{4.1}$$

The threshold value τ is then chosen randomly in:

$$[\min_{\pi \in \mathcal{P}(j)} (\pi_c(x_l, y_l) - \pi_c(x_r, y_r)), \max_{\pi \in \mathcal{P}(j)} (\pi_c(x_l, y_l) - \pi_c(x_r, y_r))] \tag{4.2}$$

Let us denote as $Q = \{p_l, p_r, c, \tau\}$ a binary test defined by Eq. 4.1. Given any patch set $\mathcal{P}_Q$, it can be splitted into two disjoint subsets $\mathcal{P}_Q^l$ and $\mathcal{P}_Q^r$:

$$\mathcal{P}_Q^l = \{\pi \in \mathcal{P}_Q | Q(\pi) > 0\} \tag{4.3}$$

$$\mathcal{P}_Q^r = \{\pi \in \mathcal{P}_Q | Q(\pi) \leq 0\} \tag{4.4}$$

Fig. 4.4 illustrates the splitting process. Then, a binary test Q can be evaluated, in terms of quality, by these two criteria:

- One criterion related to Shannon's entropy of the labels $\{+,-\}$:

$$U_e(Q) = |\mathcal{P}_Q^l| \cdot H(\mathcal{P}_Q^l) + |\mathcal{P}_Q^r| \cdot H(\mathcal{P}_Q^r) \tag{4.5}$$

  where $H(\mathcal{P}) = -\frac{|\mathcal{P}^+|}{|\mathcal{P}|}\log\left(\frac{|\mathcal{P}^+|}{|\mathcal{P}|}\right) - \frac{|\mathcal{P}^-|}{|\mathcal{P}|}\log\left(\frac{|\mathcal{P}^-|}{|\mathcal{P}|}\right)$,
  and $|\mathcal{S}|$ denotes the cardinality of the patch set $\mathcal{S}$. According to this criterion, the optimal split is done when positive and negative patches set are perfectly separated in the two subsets

- One criterion based on standard deviation of the offset vectors of positive patches:

$$U_d(Q) = \sum_{t\in\{l,r\}} \sum_{\pi\in(\mathcal{P}_Q^t)^+} (d_\pi - d_{(\mathcal{P}_Q^t)^+})^2 \tag{4.6}$$

  with $d_{(\mathcal{P}_Q^t)^+}$ the average displacement computed from all positive patches displacements. The aim of this criterion is to split the positive part $\mathcal{P}_Q$ according to their spatial position with regard to their center, and gathering positives patches localized in the same area with regard to the reference point, thus concentrating the votes casted at one leaf. We can imagine for example, patches taken at the head of pedestrian, gathered in the same subset (as all of them are located above the reference point, which is usually located at the torso level)

Both criteria are used to train non-leaf node. Indeed, at each non-leaf node j, Gall randomly selects one of these two criteria, and then draws N_Q binary tests. He then conserves the test Q_{opt}^j optimizing the selected criterion:

$$Q_{opt}^j = \underset{Q}{\operatorname{argmin}}\, U_b(Q) \tag{4.7}$$

with $b \in \{e, d\}$. Moosmann only considers one criterion, based on entropy and mutual information. Otherwise, his binary test is similar to Gall's one, with the difference that Moosman does not compare the difference of two values to one threshold, but directly compares the value of one pixel, in one feature, with one threshold value.

Then, after dividing the patch set $\mathcal{P}(j)$ by using Q_{opt}^j, $\mathcal{P}(2\cdot j+1)$ and $\mathcal{P}(2\cdot j+2)$ are trained in a similar way, if they are non-leaf node. In case of a leaf node (when the depth reaches the maximum depth, or when there are not enough positive patches), all displacements from positive patches are stored, and the leaf is associated to a weight $\omega_j = \frac{|\mathcal{P}^+(j)|}{|\mathcal{P}(j)|}$, which is the

proportion of positive patches in $\mathcal{P}(j)$. If we look at Moosmann's paper, each leaf also contains one SVM, classifying all patches according to their class (he is working on multi-class classification).

4.2.1.3 Implementation details

In the previous section, we explained how the Hough Forest can be basically trained. However, in Gall's work [GL13], several complementary details are used to enhance the detection. We present them in this section.

The first operation consists in normalizing positive image size. The positive training set is composed of different instances of the same class, at different sizes. One solution suggested by Gall was to resize all images in order to make them have the same height, by ensuring that the average value of the largest dimension of all positive images equals 100 pixels.

The second modification is done in order to add some variability within the forest, and enhance the classification of the most difficult examples. One first set of trees from the forest is trained by using the whole patch set. Then, this partial set is tested in the whole dataset, in order to extract positive images with the lowest peak, and negative ones with the highest peak. The aim is to get the subset of the most difficult positive and negative images. This subset is then used to train a second subset of trees of the forest. Finally, a third set is generated the same way.

Third, Gall defines two criteria to determine if a node is a leaf:

- If node's depth is equal to d_T
- If there are not enough positive patches as input for node training ($\mathcal{P}^+ < n_{min}$ with $n_{min} = 15$)

4.2.1.4 Detection

In the last section, we explain the detection of instances of object once the forest is trained.

Given a test image $\mathbf{I}$, all patches π are densely extracted. Gall also studied impact of drawing patches using a regular grid [GL13]. For each tree, each patch goes through the tree until it reaches a leaf node i. Then it votes according to all displacements stored in the leaf, each vote having a weight equal to $\frac{1}{|\mathcal{P}(i)|}$. Finally, a detection map (the Hough Transform) is provided, and output can be obtained by maxima search, or by Mean shift

[CRM03]. To cope with scale changes, it is possible to rescale $\mathbf{I}$ to different sizes, and perform maxima search in scale-space dimension. Fig. 4.5 illustrates some good and bad detections. Gall proposed to accelerate the

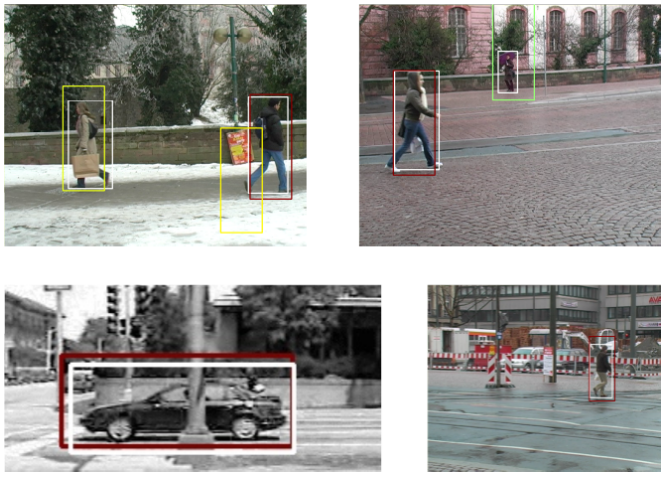


FIGURE 4.5: Failures and right detections

detection by making vote only patches that belong to leaves where the proportion of positive patches is above 0.5.

4.2.2 Extensions of the Hough Forest

Many contributions aimed to enhance the original Hough Forest. In this section, we propose a literature review of these different works.

Gall in [GRVG12] proposed an extension of his Hough Forest, to estimate the size of the detected instances, and to deal with multi-class detection (or object recognition). To estimate the size of the detected instance, Gall uses the backprojection operation [RGVG10]; [LLS08], by searching for patches which have voted for the peak. Then, by thresholding the backprojection map, he is able to estimate the size. Gall also uses this backprojection map to determine a similarity between detected instance and images from the training set. Then, to deal with multi-class detection, instead of using several forests, Gall prefers using one forest containing several classes. To do that, for each class, Gall proposes a measure of similarity between classes, based on the quantity of patches from one class present in each leaf, and on the proportion of patches from other classes in the same leaf.

In [Gal+11], Gall proposed to extend Hough Forest to tracking and action recognition. For tracking, the Hough Forest is trained offline, before

the tracking process. To cope with shape change, weights of displacements are updated using backprojection. For action recognition, more details are available in [YGVG10]. Hough Forest are built using training sequences, and by using 3D patches (associated to label related to a specific action and to a 3D displacement vector) as training set. Each patch is described using the grayscale color space, x, y and time derivatives, and optical flow. Then, for recognition, 3D patches are going through all trees, and vote for an action, and a spatio-temporal center.

Wohlhart [Woh+12] proposed a method to enhance the Hough Forest by considering a method to validate or reject possible locations x of instances. To do that, he defines an activation value for the displacement d to the location x. This activation value corresponds to the quantity of votes casted by any patch which has voted at the direction d for x: the higher it is, the more d has contributed to x (in Wohlhart's work, a patch located at y does not cast only one vote at $y+d$, but casts a vote to the neighborhood of $y+d$, followed by a Gaussian filtering of this vote). The higher the activation value is, the more it has contributed to the vote for x. With this formulation, the value of the Hough Transform at x corresponds to the sum of the activation values for x of all displacements vectors stored in the whole Hough Forest. This formulation allows him to validate or reject potential instances.

Murai [Mur+15] proposes a training enhancement, as he plans to modify weight of patches according to their appearances by reducing weights of positive patches that are too similar to negative ones. The aim of his work is to increase the influence of patches that are very different from those found on negative images.

Recently, Ciolini [Cio+15] presented a work that aims to accelerate detection, and make it runs on low-power systems. His forest is smaller than Gall's one (5 trees against 15) by adding to the training set of the tree t, some false positive obtained from the forest $\{0, 1, \ldots, t-1\}$. At each leaf, Ciolini also clusters all positive patches according to their relative displacements, in order to reduce the quantity of voting pixels. In detection step, instead of using all patches, he subsamples them. Finally, he also computes a faster approximation of the HOG features.

4.3 Our contributions

In this section, we present our work on Hough Forest. Our purpose is to investigate the potentialities of low-level features (color and differentials at different orders) for object representation. Unfortunately, moving from the full feature space proposed by Gall (HOG + derivatives + lab color space) to reduced one (derivatives + lab color space) reduces performances (loss of power of representation). However, removing HOG features also improves the speed. So, in terms of applications, we consider relevant to search for methods to enhance accuracy in small feature space.

All contributions presented are taken from [TM16]. As in the previous section, we will provide a more detailed explanation of our work. This section will be divided that way:

- First, we explain how we worked on training set generation
- Second, we explain how we chose to enhance the node training

As for the last Chapter, we will not mention parameters values in this section, but in the next Section. 4.4.

4.3.1 Patch generation

As told in Section. 4.2.1.1, Gall [Gal+11] draws patches from positive and negative images to generate the training set. Those patches are randomly drawn, following a uniform process. Like Moosmann [MNJ08], we think that drawing patch randomly, without prior information, can lead to many non useful patches (without visual structure). While he proposed a method based on adaptive saliency maps, in our case, we choose to draw a certain proportion of patches randomly (still using a uniform process), while the other proportion ($\alpha^{\mathcal{J}}$) is drawn using a mathematical measure, taken from [Lin98], called *junctionness* (Eq. 2.20), already mentioned in Chapter. 2:

$$\mathcal{J}(\mathbf{I}) = (\mathbf{I}_y^\sigma)^2 \cdot \mathbf{I}_{x^2}^\sigma - 2 \cdot \mathbf{I}_x^\sigma \cdot \mathbf{I}_y^\sigma \cdot \mathbf{I}_{xy}^\sigma + (\mathbf{I}_x^\sigma)^2 \cdot \mathbf{I}_{y^2}^\sigma \tag{4.8}$$

It is computed at different scales ($\sigma_i^{\mathcal{J}}$), and at each scale, the same number $n^{\mathcal{J}}$ of patches is drawn.

These multi-scaled junction maps are computed before the forest training, and patches are obtained by using a maxima search. Each time a patch is drawn, a maxima suppression is computed around the patch center, and

this suppression is maintained during the whole training. The aim is to avoid using the same object part to train different trees, and keeping a certain variety of patches in the training set. Fig. 4.6 shows examples of strong junctions, the radius corresponding to the scale, and the color to the absolute value of junction-ness (the red corresponding to the highest values). The aim of this step is not only to draw patches with structure (in that case, a saliency map based on gradient may be sufficient), but to get structures potentially representative of the target (as we can see in Fig. 4.6, head and foot have high junctionness values). We also choose to keep a certain proportion of patches drawn randomly in order to keep a variability on the training set. Other measures can also be tested, such as blob measure [Lin98] or keypoints, but we consider that studying the impact of junction measure is sufficient. Indeed, at short-term, the purpose is not to find the best measure to draw patches, but rather to see if drawing a certain quantity of patches in a deterministic way can provide some benefits.

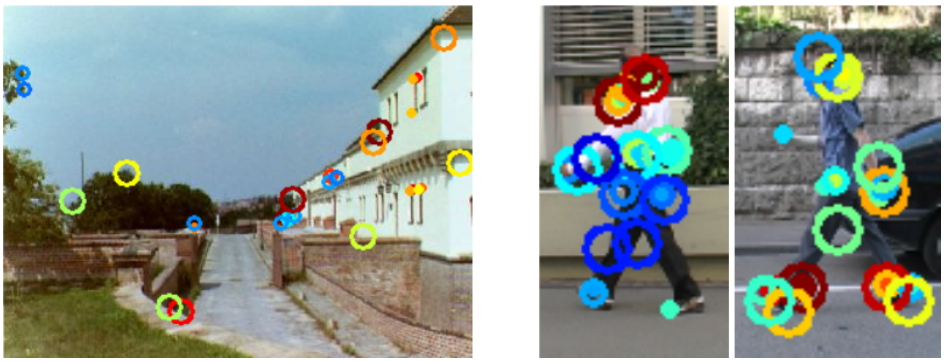


FIGURE 4.6: Junctions shown by circles for one landscape and two pedestrian images

4.3.2 Node training

The second contribution from [TM16] concerns nodes training. Each node is trained by generating N_Q binary tests, and retaining the best one (according to Eq. 4.5 or Eq. 4.6). Each potential binary test is generated using 6 random uniform processes (4 for coordinates of the left and right members of Eq. 4.1, 1 for the channel, 1 for the threshold). As for training set generation, we aim to add some prior information in the way to draw each parameter, by considering the input used to train a non-leaf node. With the parameter set given by Gall, based on patch size of 16×16 pixels and 32D feature spaces, generating a binary test $Q = \{p_l, p_r, c, \tau\}$ leads

to choose a binary test among the $32 \times 16^{2^2} = 2\,097\,152$ possible binary tests, without considering the threshold τ. From this remark, we aim to construct probability laws to draw each parameter of the binary test, using the patch set taken as in input, to reduce the quantity of generated binary tests (and then to reduce the training time).

Let us consider a node to train, indexed by i, and let us also define $\mathcal{P}_Q = \mathcal{P}(i)$ the patch set serving as a training input. Let us also consider a disjoint subdivision of $\mathcal{P}_Q$ into two patch set $\mathcal{P}_Q^{\square}$ and $\mathcal{P}_Q^{\blacksquare}$ (Both symbols $\square$ and $\blacksquare$ will be instanciated later, according to the criterion to optimize, entropy vs. spatial deviation).

The first step consists in generating global patches (that will be called *superpatches*) summarizing all information stored about $\mathcal{P}_Q^{\square}$ and $\mathcal{P}_Q^{\blacksquare}$.

For any patch π, let $\hat{\pi}$ be the patch obtained by binarizing π using Otsu's threshold [Ots75], at each channel. The aim of $\hat{\pi}$ is to provide contrast invariance, by basically indicating, for each feature, which part of the patch belongs to a high or a low level. Then, for $a \in \{\square, \blacksquare\}$:

$$\Pi^a = \sum_{\pi \in \mathbf{P}_T^a} \widehat{\pi} \tag{4.9}$$

For a given label a, Π^a can be interpreted as a probability map indicating, for each feature, which pixel is more likely to be a "upper" (resp. "lower") pixel for all patches from $\mathbf{P}_Q^a$. Fig. 4.7 illustrates construction of Π^a (the patch set on the left represented by the green rectangle corresponds to $\mathbf{P}_T^a$). However, directly using the normalized Π^a as probability density, may turn the process too deterministic, because the dynamics of Π^a can be very high. We choose instead to drastically quantize the density by using the binary superpatch $\widehat{\Pi}^a$, resulting from the threshold operation of Π^a by 0 and valued in $\{-1, +1\}$.

Finally, for a binary test Q, and a prior partition $\mathbf{P}_Q^{\square}$ and $\mathbf{P}_Q^{\blacksquare}$, we define the two superpatches Π_Q^l and Π_Q^r as follows:

$$\Pi_Q^l = 1 + \frac{1}{2}(1 + \widehat{\Pi}^{\square}) + \frac{1}{2}(1 - \widehat{\Pi}^{\blacksquare}) \tag{4.10}$$

$$\Pi_Q^r = 1 + \frac{1}{2}(1 + \widehat{\Pi}^{\blacksquare}) + \frac{1}{2}(1 - \widehat{\Pi}^{\square}) \tag{4.11}$$

Π_Q^l and Π_Q^r are superpatches valued in $\{1, 2, 3\}$, whose purpose is twofold:

- For a given feature channel, normalized Π_Q^l (resp. Π_Q^r) is used as a

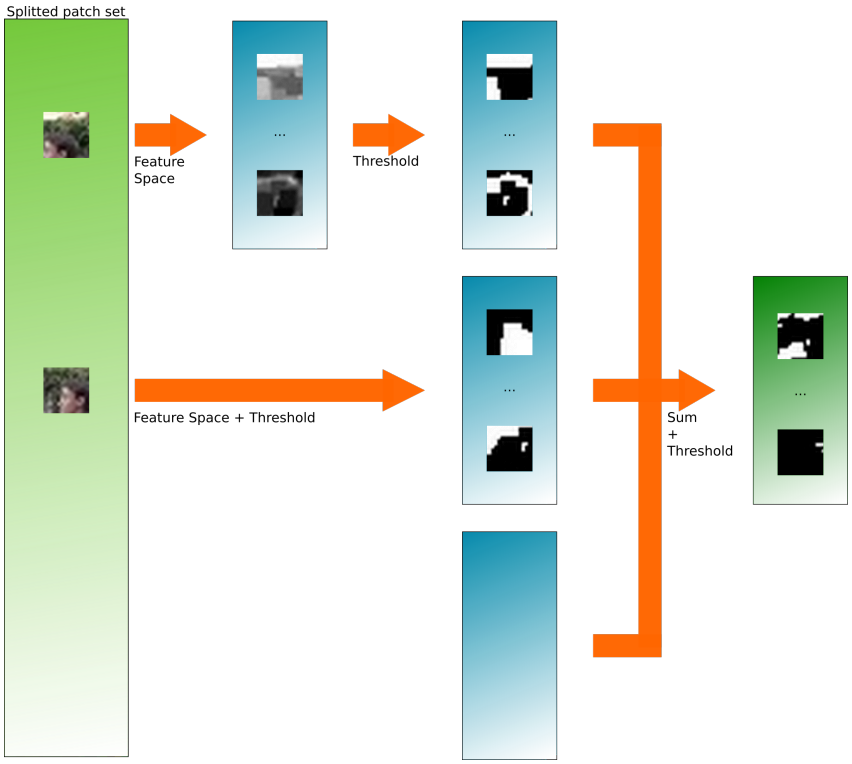


FIGURE 4.7: Generating Π^a

probability map to draw the left (resp. right) pixel member p^l (resp. p^r) used in Eq. 4.1.

- To select the most relevant feature channel, the (normalized) sum of the standard deviations of Π^l_Q and Π^r_Q is used as a probability map to draw the feature channel.

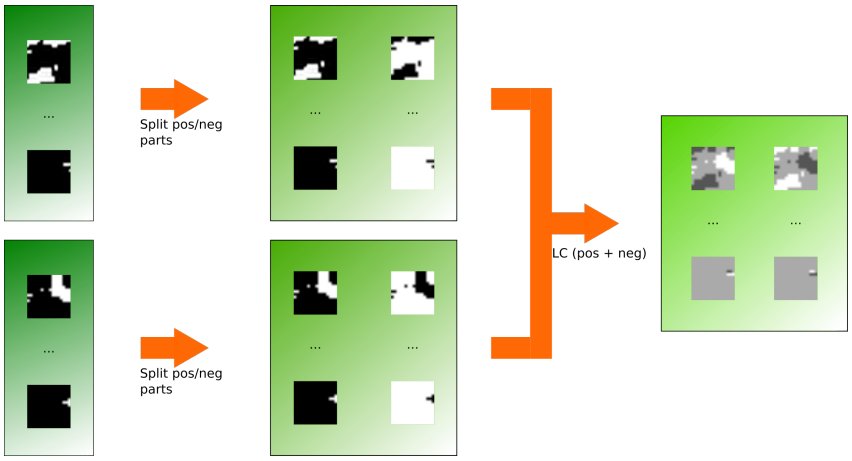


FIGURE 4.8: Generating superpatches

Fig. 4.8 schematizes the construction of one superpatches (LC stands for Linear Combination, and represents Eq. 4.10 and Eq. 4.11). Finally, the

parameters $\{p^l, p^r, c\}$ are drawn according to the probability map defined before.

Now the missing step consists in determining the two subsets $\mathbf{P}_Q^{\square}$ and $\mathbf{P}_Q^{\blacksquare}$, depending on the criterion to minimize:

- When the entropy criterion is chosen, $\{\square, \blacksquare\} = \{-, +\}$, meaning that we are simply using the labels from negative and positive example patches of $\mathbf{P}_Q$
- With the spatial deviation criterion, considering the Eq. 4.6 to minimize, we apply a k-means (with $k = 2$) clustering of the patch set $\mathbf{P}_Q^+$ with respect to the displacement vector d_π. Indeed, the k-means algorithm is obviously a better way to minimize the spatial deviation criterion than Gall's random method. In this case, $\mathbf{P}_Q^{\square}$ and $\mathbf{P}_Q^{\blacksquare}$ are the two sets resulting of the 2-means of $\mathbf{P}_Q^+$

One element we did not studied was the selection of the threshold τ.

4.4 Experiments

In this section, we present some results obtained from academic datasets. In particular, we will work on two datasets: UIUC-cars [AAR04] and TUD-Pedestrians [ARS08]. Before detailing characteristics of these two datasets, we will first present the usual protocol to evaluate detectors.

4.4.1 Evaluation method

In this section, we present methods of evaluation, used in the two datasets. As for tracking, let us use the formulation $B = \{c, w, h\}$ to define a rectangle of center c, and dimension $w \times h$. Following Gall's method to resize all objects, let (w_{ref}, h_{ref}) be the characteristic size of the class.

Given an image $\mathbf{I}$, considering GT_i an annotated instance of the class to detect (represented by its bounding box), GT_i is considered detected if the detector gives as an output a bounding box B such as $O(GT_i, B) \geq 0.5$, with O the overlap measure already defined in Section. 3.4 (Eq. 3.29). From this notion, four types of detection can occur:

- True Positive (TP): an instance rightfully detected
- False Negative (FN): a non detected instance

- False Positive (FP): a detected instance for a background area
- True Negative (TN): an instance of the background class rightfully non-detected

These four measures can be used to evaluate detectors. However, these criteria are not normalized. Moreover, in detection context, the notion of True Negative is not relevant: every background area classified as a background is a True Negative. Fig. 4.9 illustrates TP (in blue), FP (in red) and FN (the pedestrian on the left) in one frame from TUD-pedestrians [ARS08]. The ROC curve space, more suitable as it contains more in-

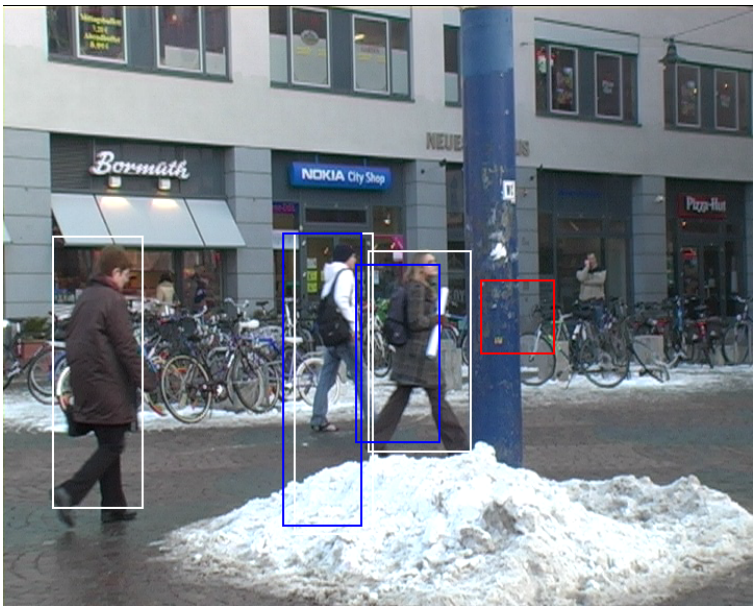


FIGURE 4.9: Examples of TP, FP and FN

formation, is preferred. It is defined by two measures valued in $[0,1]^2$, coming from those defined above:

- Precision defined by $PR = \frac{TP}{TP+FP}$
- Recall, defined by $RC = \frac{TP}{TP+FN}$

ROC curves are generated by considering the trained detector and a parameter t, and by plotting the curve $(1 - PR(t), RC(t))$. We can extract relevant informations from ROC curves, such as the equal-error rate (EER), defined by a $FPR = TPR$. Another usual measure of the quality of the detector is the area under the curve (AUC).

For Hough Forest case, to generate those ROC curves, we first compute the total number of TP, FP and FN from the whole dataset, at a

fixed parameter t. This parameter t is a threshold between 0 and 1. For one image $\mathbf{I}$, Hough Forest is computed at different scales (σ_i), leading to a family of Hough Transform $\mathbf{HT}_i(\mathbf{I})$. In Gall's original work, Hough Transform is also computed at different aspect-ratios. But, in our case, as we plan to study impact of our work on the Hough Forest, and not enhance performances of the tracker, we will only work with one aspect ratio. Those Hough Transforms are then normalized by $\max_p(\mathbf{HT}_i(\mathbf{I})(p))$, to have all values of the Hough Transform lower than 1.0. Then, localization at a threshold t consists in locating all parameters (p, σ) such that $\mathbf{HT}_\sigma(p) > t$. To avoid multiple counts of the same instance, we proceed to a maxima suppression, by setting, for all σ and all $x \in (\sigma \cdot p, \sigma \cdot w_{ref}, \sigma \cdot h_{ref})$ in HT_σ), $HT_\sigma(x) = 0$. Finally, given an annotated ground truth GT and a potential detected instance $R = (\sigma \cdot p, \sigma \cdot w_{ref}, \sigma \cdot h_{ref})$, GT is detected if $O(GT, R) > 0.50$, with O the overlap measure defined in the previous chapter (see Eq. 3.29). In our case, we also consider that there are at max 5 instances per image.

The protocol described above is used in all our experiments. However, some minor details (in terms of parameters) will be specifically used for the different datasets. Parameters used for our tests are summarized Tab. 4.1.

Parameter	Gall	Ours
Training parameters		
Forest size	15	9
Depth max of each tree	15	15
Number of positive patches	25000	25000
Number of negative patches	25000	25000
Number of Potential tests per node	20000	500
Minimum number of positive patches	20	20
Proportion of patches drawn deterministically		0.30
Detection parameters		
Spatial smoothing kernel size	3	3
Scale smoothing kernel size		0.05

TABLE 4.1: Parameters set proposed by Gall, and ours

4.4.2 UIUC Cars

The first dataset we worked on is the UIUC-Cars dataset [AAR04]. The aim is to train a car (viewed from the side) detector. The training set is

composed of 550 positive images showing cars at approximately the same size (100×40 pixels), and 500 negative images. Fig. 4.10 illustrates images from the training set. To test the detector, two datasets are available:

FIGURE 4.10: Negative and positive images from UIUC-Cars

- One composed of 170 images containing 210 cars, all with the same size as in the training set
- One composed of 108 images containing 138 cars at different sizes

In all cases, images are in grayscale. Some cars suffer from low illumination or occlusion.

Gall tested his algorithm with a reduced feature space, by discarding color ones and HOG ones. After some tests, this dataset has revealed to be too simple to study impacts of our contributions: we just show the ROC curve Fig. 4.11. We tested only three situations: detection with only the first order derivative (FirstDer), detection with first and second derivatives (SecondDer), detection with first and second derivatives and 10% of patches drawn using the junction-ness measure (SecondDer + Jun). Dashed line corresponds to the EER curve.

4.4.3 TUD Pedestrian

The second dataset we worked on is the TUD Pedestrians dataset [ARS08]. Both training and testing sets are composed of colored images. All pedestrians are viewed laterally. The training set is composed of 400 images (originally 210 before updating) representing a pedestrian walking in an urban area. Each image is associated to a segmentation map, delimiting the foreground to the background (see Fig. 4.12). The test set is composed of 250 images, showing pedestrians at different sizes, and walking on different planes orthogonal to the camera.

In Gall's computation, it was decided to extend the training set by using images from the INRIA-Person dataset [DT05]. The problem tackled by Gall was the lack of background environment in the TUD-Pedestrians

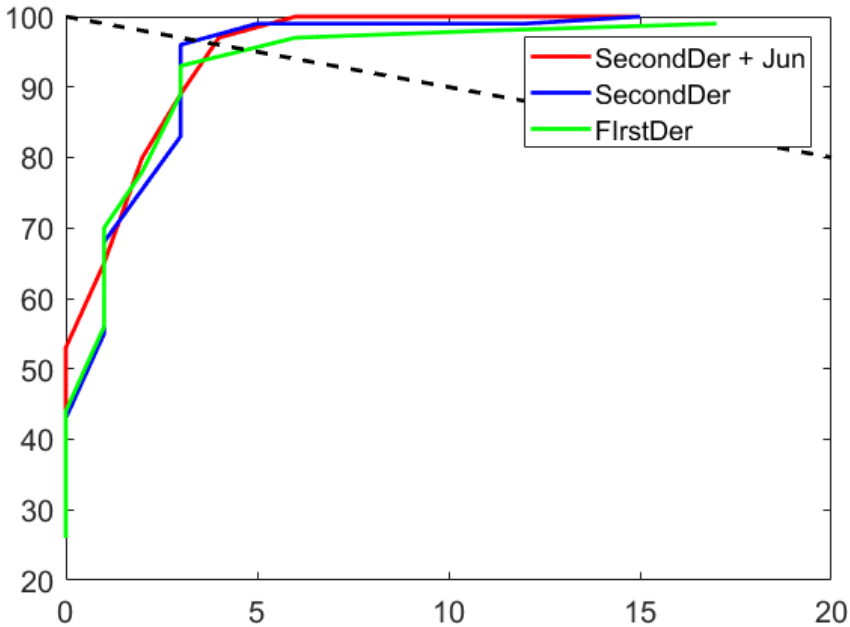


FIGURE 4.11: ROC curve on UIUC Cars multi-scale dataset

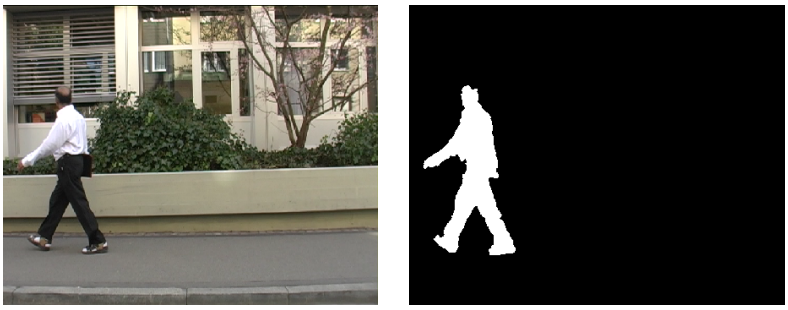


FIGURE 4.12: One training image from TUD-pedestrian with its associated segmentation map

dataset. For all tests (even the original Hough Forest), we used our own implementation. However, we took Gall's implementation of HOG features [1]. To deal with multiple sizes, each tested image is resized to different scales $\{0.3, 0.4, 0.5, 0.6\}$ before any detection.

We tested many hypotheses:

- Original Hough Forest [GL13] with our set of parameters Tab. 4.1, denoted GallHOG
- Original Hough Forest without HOG features (color and derivatives only) . The gradient will be computed using Sobel filter (Sobel)

[1] https://pages.iai.uni-bonn.de/gall_juergen/projects/houghforest/houghforest.html

- Hough Forest without HOG features, but with derivatives computed at different scales (1 and 2) (Derivatives1 and Derivatives2)
- Hough Forest without HOG features, with multi-scale derivatives (scales 1 + 2) (MSDerivatives)
- Hough Forest without HOG features, with multi-scale derivatives and our first contribution (30% of patches drawn using junctionness measure, computed at scales $\{1, 2\}$) (MSDerivatives + Jun)
- Hough Forest without HOG features, with multi-scale derivatives and second contribution (node enhancement) (MSDerivatives + SP)
- Hough Forest without HOG features, with multi-scale derivatives and the two contributions (MSDerivatives + Both)

In all cases, we still apply the min and max (erosion/dilation) operations.

GallHOG serves as a reference. Fig. 4.13 represents ROC curves of GallHOG (in blue), Sobel (in red) and MSDerivatives (in orange). All curves are obtained by computing all types of forest using parameters detailed in Tab. 4.1. For each hypothesis, 10 forests are trained, and results displayed Fig. 4.13 are the average ROC curves of all hypotheses. This average is done by averaging precision and recall obtained for each threshold values. The comparison Gall/Sobel (blue vs. red) demonstrates the usefulness of HOG features, and also relevance of our work: Sobel alone is less accurate than Gall. Indeed, in terms of EER, Gall's EER is equal to 85%, while Sobel gives 74%. However, in terms of training and detection time, Sobel is much faster: 30 minutes to train the whole forest (against 1 hour for Gall), and detection made on the whole test set (250 images) takes 15 minutes for Sobel (3 seconds per image), 22 minutes for Gall (4 seconds par image). Moreover, in terms of memory consumption, we move from a $16 \times 16 \times 32$ pixels to represent a patch for Gall, to $16 \times 16 \times 14$ for Sobel, saving more than 50% of the memory. Now, comparing Gall and MSDerivatives (blue against orange), MSDerivatives is slightly below Gall in terms of AUC, but is still better than Sobel. EER is also close to Gall's EER (81%). The training time is reduced by 2 (30 minutes) while detection time is reduced by one third (about 15 minutes for 250 images, so 3 seconds per image). In terms of area under the curve, GallHOG clearly performs better than MSDerivatives. If we consider the point from the ROC curve obtained with a threshold of 1 (the detector only

detects the highest peak from the multi-scale Hough Transform), the one obtaind from GallHOG is better than the one from MSDerivatives: 98% against 92%. In terms of application, if both detectors are configured to detect only one pedestrian, GallHOG will perform fairly better. Globally, those results demonstrate interest of multiscale derivatives as an alternative to HOG features.

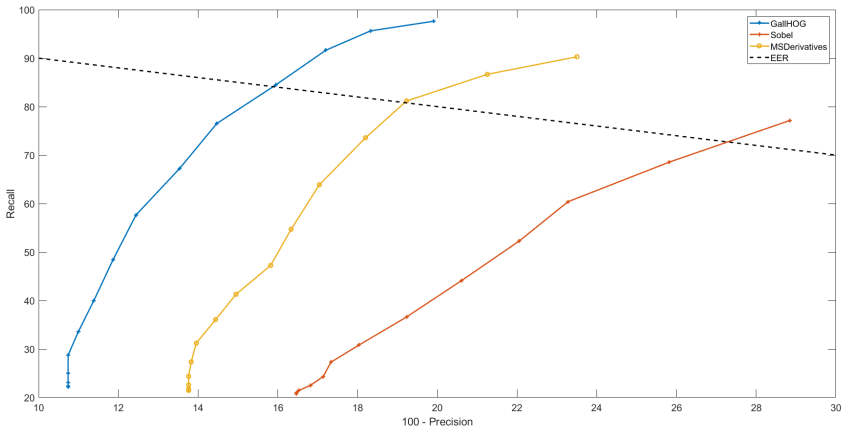


FIGURE 4.13: GallHOG *vs* Sobel *vs* MSDerivatives

The main contribution of [TM16] concerns the stability of the detector. To test this, as indicated Tab. 4.1, we reduced the forest size (15 trees *vs* 9) and the number of potential nodes generated ($20\,000$ *vs* 500). In these conditions, we can first notice how close we are to the average ROC curve obtained with Gall in its original paper [GL13], with the parameter set proposed by Gall (second column of Tab. 4.1). In both cases, EER is 86.5%. However, with the small number of potential nodes, performances will widely vary. Then, as we want to study the stability of detector's performances with respect to the characteristics, we run 10 rounds of training and test in the dataset. We obtain two results. First, Fig. 4.14 was obtained by considering four hypotheses: MSDerivatives, MSDerivatives + Jun, MSDerivatives + SP and MSDerivatives + Both, and plotting the worst and the best curves in the three cases (in terms of area under the curve). As we can see, using a saliency map to draw a certain number of patches (30% in our case) significantly improves the stability (results in red), compared to the min and max curves of MSDerivatives (in blue). Adding the superpatch contribution increases the gap between those (green curves), and seems to reduce the role of the saliency map (as MSDerivatives + Both and MSDerivatives + SP produce similar min max curves). In the

three cases (Jun, SP and Both), results are still slightly more stable than the original method.

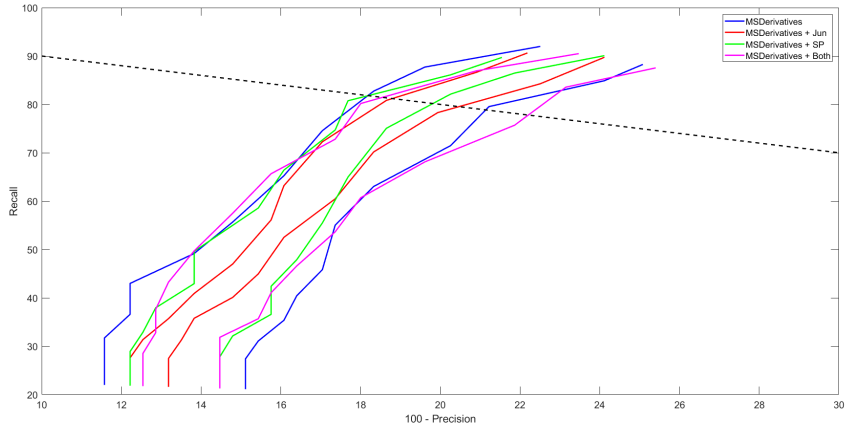


FIGURE 4.14: Worst and best curves in different cases

The second results we get are shown Fig. 4.15. We took the same set of results as Fig. 4.14, but instead of looking for the best and worst curves, at each threshold value used to compute all curves, we compute the covariance matrix, and represent it by an ellipse (slanted according to eigen vectors and with size proportional to eigen values). All curves are obtained by averaging all the 10 ROC curves obtained at each series. In this case, superpatch impact is more relevant, as ellipses are globally smaller than those originally obtained (it is important to note that the dynamic of ordinate axis is higher than abscissa, and projection on this axis shows that the simple MSDerivatives in blue is less stable).

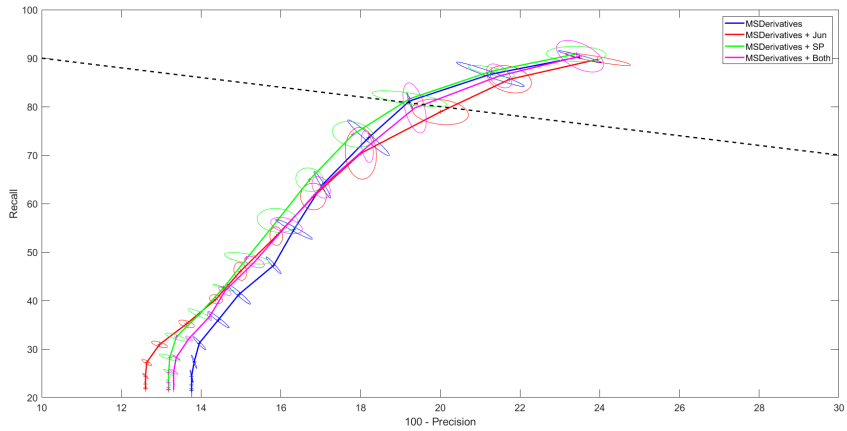


FIGURE 4.15: Average ROC curves, and covariance ellipses for different threshold points

One last study concerns the impact of spatial derivatives. We saw Fig. 4.13 that the detector is performing much better moving from Sobel based to multi-scale derivatives. But, what is happening when we are still working with one space derivative? In that case, we compared three hypotheses: MSDerivatives (scales 1 and 2), DerivativeS1 and DerivativeS2. As the patch size is equal to 16, it is not relevant to work on larger scales. Average ROC curves (obtained from 10 realizations) are displayed Fig. 4.16. It is interesting to note that moving from Sobel based to Gaussian derivatives leads to a better detector. The second remark is that DerivativeS2 and MSDerivatives are very similar. In that way, and as DerivativeS2 is lighter to compute and train, we may prefer exploiting this feature space.

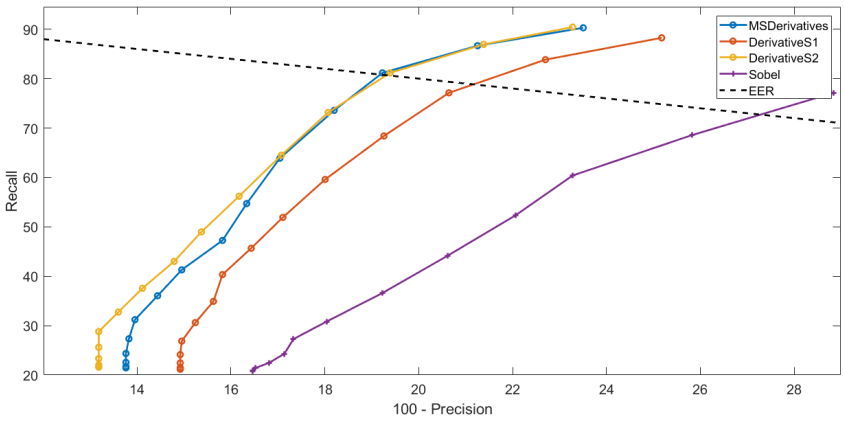


FIGURE 4.16: MSDerivatives *vs* DerivativeS1 *vs* DerivativeS2 *vs* Sobel

4.5 Conclusion

This section, dedicated to object detection, was aimed to determine impact of discarding HOG features from the original Hough Forest [GL13], to work only with (multiscale) derivative features. Results on Section. 4.4.3 showed that the obtained detector was slightly less accurate than [GL13], but still gives decent results, with smaller computation time and memory footprint. However, we are still below real-time.

Our results in object detection are then not fully satisfactory. Furthermore, our experiments were done on relatively outdated datasets The next objective will be to test it on harder datasets, such as those proposed by Gall (INRIA persons [DT05] involving human with different poses, or

Weizmann horses [SBC08] requiring votes in the aspect ratio parameter). At longer term, the objective will be to apply the detecor on multi-class detection, and test it on related datasets (Pascal VOC challenges notably [Eve+07]; [Eve+10]).

In terms of contributions, [TM16] does not show major improvements in terms of accuracy. However, some interesting elements can be studied. In the patch drawing, using a geometrical measure (junction-ness) to draw some patches leads to a more stable ROC curves. The next question would be naturally to test other saliency measures (such as blob-ness [Lin98]) and maybe keypoints. Concerning the superpatch approach, the final detector is slightly more stable (see Fig. 4.15), but results are still relatively limited. It may be due to the fact that superpatches are only valued in $\{1, 2, 3\}$, and are built from binary patches: leading to a non significant impact in terms of precision or stability. Our aim was to limit impacts of strong peaks. However, it seems that it limits advantages of our approach. One solution may be to increase the dynamic of the superpatches. Another remark concerns Eq. 4.1: superpatches approached was aimed to draw all parameters except τ by a specific probability law. However, this last parameter τ can degrade the quality of the binary test, as it is chosen randomly according to a uniform law on Eq. 4.2. In that way, especially when the range defined by Eq. 4.2 is large, the weak probability to draw a relevant value for τ can lead to a useless binary test, even though other parameters are drawn correctly. We will then study methods to improve the choice of τ in the future.

If we want to combine our contribution with other Hough Forest extensions (see Section. 4.2.2), Wolhart's work [Woh+12] may be combined with the junction-ness measure. Indeed, on the one hand, his notion of activation value depends on the positive patch's displacement, and is higher when the displacement has well contributed to a vote. On the second hand, the junction-ness measure is able to draw patches in some relevant areas (such as pedestrian's foot). If we suppose those patches's displacements very similar (most of the time, people's feet are at the lower part of the body), we can suppose that naturally, those displacements will have high activation values in Wohlhart's definition. In that way, combining [TM16] and [Woh+12] may improve results. Similarly, we can also think about combining the junction-ness measure (or other geometrical measure) to works of Murai [Mur+15], as he aims to increase the weight of positive patches visually different from negative patches.

Regarding perspectives related to other extensions, we can refer to [Gal+11] to extend Hough Forest with derivatives features for other applications. For tracking context, our work on tracking [TM15]; [TM17] showed that it is still possible to exploit the GHT in its simplest form for accurate tracking. For action recognition, the question is interesting, as temporal features (time derivatives and optical flow for instance) seem essential for recognition. This question should be tackled in the future. Similarly, works of Razawi about backprojection map [RGVG10] should also be studied in the future.

The next and final chapter of the thesis will open the perspectives of our works on tracking and detection.

Chapter 5

Perspectives and Conclusion

This chapter closes the thesis. It will be divided into two parts. In the first part, after recalling the aim of our work, we will summarize all contributions presented in this thesis. We already dealt with perspectives related to object tracking and detection independently. The second section will be dedicated to further discussions on the possible implementation of our algorithms in a low-cost system, and the related perspectives in terms of autonomous systems.

5.1 Conclusion

We aimed to study benefits and limits of using exclusively pixel colors and scaled derivatives, spatially pooled by the Hough Transform, for two applications: object tracking and object detection.

The first Chapter was dedicated to image representation. A review of color-based and shape-features, detailing how, from low-level features, higher level ones have been developed: color attributes [VDW+09] or HOG [DT05] notably. Due to the importance of the Hough Transform, we dedicated the second part of the chapter to it, recalling its history, presenting its numerous variants, and notably the Generalized Hough Transform. We have emphasized its versatility, by mentioning its use in many applications.

Our work on object tracking was presented in the second chapter. Given different types of tracking, we started by setting our framework and explaining challenges related to tracking. Then, after a literature review presenting state-of-the-art, real-time and Hough based trackers, we presented our contributions, starting from a tracker combining the original GHT with a Particle Filter [TM15] and ending with a tracker substituting to the Particle Filter a simple color histogram [TM17]. The last tracker has proven effective and has been tested and evaluated on VOT14 and VOT15

datasets. The second dataset has been used for further studies of [TM17]: parameters tuning and feature space studies. This tracker was sent to the VOT committee, for an active participation on the VOT17 challenge. It has been accepted, and final ranking will be announced in October during the VOT2017 workshop.

Object detection is the subject of the third Chapter. After formalizing the problem, we presented some state-of-the-art and Hough-based detectors. Then, as our contribution is strongly related to the Hough Forest [GL13], we presented Gall's work and Hough Forest extensions. Then, we detailed our contributions, aiming to improve the stability of the detector. Finally, a section dedicated to experiments presented results on UIUC Cars dataset and TUD pedestrians. Experiments showed that reducing the feature space originally proposed by Gall (by discarding HOG features and keeping only derivatives and pixel colors) presents a real interest.

Regarding our working hypotheses, in tracking, the outcome is satisfactory: we got a fast but effective tracker relying on local features, with light functions (color histograms, and GHT used in its purest form), which has proven competitive (good results on VOT14 and 15, participation to VOT17). Concerning detection, results are more mitigated: outcome from [TM16] is somewhat disappointing, but we have shown that scaled derivatives can compensate the lack of HOG features. However, results obtained from [TM16] are not as satisfactory as we hoped.

One concrete application of our work is the implementation on an autonomous system. The final section of the thesis deals with discussions on the implementation of our work on a low-cost system. We will also consider our work in a global point of view, to give possible perspectives made by combining the Hough-based tracker and detector.

5.2 Perspectives

This section will be divided into two parts:

- First, we will mention implementation of the studied tracker and detector on Raspberry Pi 3. The interest is to provide a better understanding of what is necessary to implement an autonomous system based on our work

- Second, we will take a global point of view, and explain how to combine the tracker and the detector for higher level vision tasks

Perspectives presented in this chapter are at middle to long-term, compared to those presented on chapters dedicated to tracking (Section. 3) or detection (Section. 4).

5.2.1 Raspberry Pi 3 Implementation

The Raspberry Pi is a single-board computer originally designed for educational purpose. Different enhancements in terms of hardware led to the Raspberry Pi 3 (equipped with an ARM processor), which is now used for different purposes (digital media player, robotics, home automation...). A picture of the computer is available Fig. 5.1 (photo taken from [1]). Given

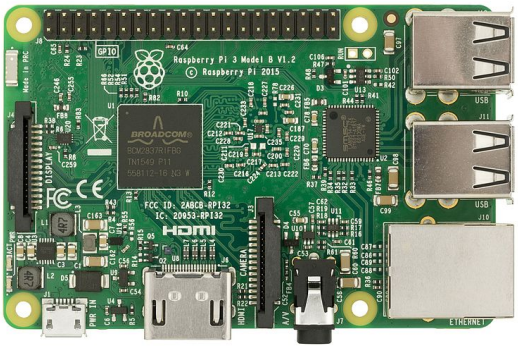


FIGURE 5.1: Raspberry Pi 3 Model B

its size (credit card format), its low cost and the availability of RGB and thermal camera, we used it as a support to study behavior of our work, and more generally, to study computer vision algorithms on low cost systems.

The tracker has been implemented under its two forms: the "position tracker only" and the "position + scale" versions. From a sequence directly acquired by the camera (webcam or Raspberry Pi specific camera), the user can select one or several objects to track. The first aim of the test was to measure the speed of the tracker on low-cost system. The test was successful, since, except for big objects, tracking can be done at real-time (more than 20 fps), even in multiple targets mode (at most 5 targets). One weakness found in VOT15 was the relative inability of the "position + scale" tracker to correctly estimate the target scale. Testing our tracker on live sequence confirm this issue. We are able to test our tracker in diverse situations (environment, context) that we can control (illumination), by selecting any kind of target. Due to the size of the computer, we can

[1] https://en.wikipedia.org/wiki/Raspberry_Pi

consider embedding the Raspberry Pi on a small vehicle, to experiment some specific camera motion. From an academic point of view, in contrast with evaluation on public datasets, which always present some bias (type of objects, difficulties related to chosen sequences), the interest is the possibility to test and evaluate our tracker on any setup.

The Hough Forest detector has also been implemented on Raspberry Pi, but only in its original version (with HOG features). We used forests trained on TUD Pedestrians. In this condition, with the Raspberry camera, we are able to run the detector at about 10 fps, by rescaling the image, and computing the Hough Transform in only one scale. The detector is then able to detect people measuring 1.80m at about 3m from the camera. This limitation can be compensated with some explicit multi-threading (each thread computing the Hough Transform at a different scale). This solution, however, adds a constraint in terms of hardware (multi-core processor). We did not test the detector with derivatives features only. This version should be faster, but since moving from Derivatives + HOG to Derivatives only reduces the detection time of about 33%, real-time should not be achieved anyway.

5.2.1.1 Global perspectives

Previously, we used both tracker and detector implementations on Raspberry Pi 3 to elaborate perspectives of the two algorithms taken separately. This section is dedicated to mutual benefits. Indeed, as both algorithms are using derivatives features (gradient only for [TM16] and scaled derivatives for [TM17]) and are unified by the Generalized Hough Transform (in its purest form for one, and with a Random Forest replacing the R-Table for the second), combining the two algorithms for diverse purposes is natural.

Using the tracker as a base, and the Hough Forest as a support can lead to object-specific tracker, when pre-trained (pedestrian tracker for example). Inspired by [GRB13], a more interesting task can be to train the Random Forest (or the Random Fern in Godec's case) online to move from our current short-term tracker, without recovery function, to long-term tracking, like TLD [KMM12]. In full occlusion case (pedestrian walking behind a pillar for example), our current tracker will fail, and the necessity to compute the foreground/background color model could be very time consuming for recovering the target. Using the GHT alone on the whole scene

until recovery can be a solution (with a threshold on the peak for the recovery), but may not be effective (since the GHT alone does not perform well on VOT2015). In that case, a Hough Forest, trained online may give better results. The forest training can be similar to Godec's method [GRB13]: all binary tests can be generated offline, while the forest is trained online, and updated every frame: at each frame, positive and negative patches are drawn according to target state, and are going through all (already) generated trees to build the set of displacements stored in all leaves. As we are only computing derivatives both in tracking and detection and the tracker is very light, we should be able to do online training while remaining real-time. The real difficult problem would be to propose a method to switch from one algorithm to the other. More precisely, we have to estimate failure case for tracking (using a combination of Bhattacharyyaa coefficient and variation measure of the Hough peak), and validate recovery for the detector (Hough peak only for instance).

Conversely, with a detector as a base and a tracker (or several instances of the tracker) as a support, we can address object counting . A detector, combined with a simple prediction model or a labeling, can do it. Indeed, a detector alone may count and recount the same instance. However, currently, our detector is still relatively heavy in terms of computation time. A more effective solution involving a tracker should be to create an instance of tracking for each new target detected by the detector. Then, by not performing detection on the areas of tracked targets, recounting issues can be avoided. Moreover, reducing the detection search area limits the time consumed by the detector, which should be the heaviest function of the system. An involved problem is the robustness to noise of the tracker: in Chapter. 4, a correct detection is considered for overlap measure (Eq. 3.29) above 0.50. In this case, the tracker can be badly initialized. Even though results on VOT14 showed a certain robustness to noise (see Tab. 3.4), disturbed initialization as tested in VOT14 may be very moderate compared to those involved by a poor detection.

In any case, these two perspectives have an interest in terms of applications for autonomous systems. Benefits are related to low computation time, due to a unique and light feature space (color + spatial derivatives only) and a low memory footprint (Generalized Hough Transform as a main algorithm, tracker model consuming less than 50 ko), making the hypothetical autonomous system usable on embedded or low-cost conditions. However, in terms of accuracy, we need to realize further studies.

One major limit concerns the scale estimation in both cases. In tracking case, our implemented solution is not satisfactory [TM17]. In detection case, multiple scales (and multiple ratios) Hough Transform may give decent results, but is very slow. One solution to further investigate are the backprojection maps, since Razavi [RGVG10] and Gall [GRVG12] demonstrated that this operation can be used for different purposes (multi-view detection, bounding box estimation...).

Appendix A

Per sequence results on VOT2015 for different color spaces

	[TM17]		ab		lab		Grayscale	
Sequence	Ove.	Fail.	Ove.	Fail.	Ove.	Fail.	Ove.	Fail.
bag	0.33	0.00	0.31	0.00	0.28	0.00	0.28	0.00
ball1	0.76	2.00	0.35	7.00	0.70	6.00	0.70	6.00
ball2	0.73	3.00	0.00	3.00	0.70	3.00	0.70	3.00
basketball	0.52	0.00	0.34	9.00	0.52	3.00	0.52	3.00
birds1	0.45	7.00	0.43	5.00	0.48	3.00	0.48	3.00
birds2	0.55	0.00	0.28	2.00	0.37	2.00	0.37	2.00
blanket	0.48	0.00	0.53	1.00	0.65	1.00	0.65	1.00
bmx	0.21	0.00	0.10	0.00	0.18	0.00	0.18	0.00
bolt1	0.48	0.00	0.52	0.00	0.46	0.00	0.46	0.00
bolt2	0.53	0.00	0.55	1.00	0.50	0.00	0.50	0.00
book	0.16	9.00	0.11	5.00	0.18	8.00	0.18	8.00
butterfly	0.44	1.00	0.42	1.00	0.41	1.00	0.41	1.00
car1	0.67	3.00	0.06	65.00	0.65	2.00	0.65	2.00
car2	0.76	0.00	0.78	0.00	0.79	0.00	0.79	0.00
crossing	0.50	1.00	0.48	1.00	0.50	1.00	0.50	1.00
dinosaur	0.41	1.00	0.27	5.00	0.36	4.00	0.36	4.00
fernando	0.37	1.00	0.41	3.00	0.39	2.00	0.39	2.00
fish1	0.35	4.00	0.19	8.00	0.37	5.00	0.37	5.00
fish2	0.22	8.00	0.19	9.00	0.22	8.00	0.22	8.00
fish3	0.44	0.00	0.22	7.00	0.46	2.00	0.46	2.00
fish4	0.28	0.00	0.32	2.00	0.35	2.00	0.35	2.00

	[TM17]		ab		lab		Grayscale	
Sequence	Ove.	Fail.	Ove.	Fail.	Ove.	Fail.	Ove.	Fail.
girl	0.63	1.00	0.52	7.00	0.47	2.00	0.47	2.00
glove	0.45	3.00	0.10	9.00	0.43	2.00	0.43	2.00
godfather	0.51	0.00	0.47	0.00	0.48	0.00	0.48	0.00
graduate	0.34	5.00	0.38	6.00	0.42	5.00	0.42	5.00
gymnastics1	0.42	8.00	0.38	7.00	0.40	9.00	0.40	9.00
gymnastics2	0.58	2.00	0.61	6.00	0.65	6.00	0.65	6.00
gymnastics3	0.31	4.00	0.19	4.00	0.31	4.00	0.31	4.00
gymnastics4	0.42	3.00	0.35	4.00	0.32	5.00	0.32	5.00
hand	0.36	9.00	0.29	9.00	0.30	9.00	0.30	9.00
handball1	0.53	4.00	0.23	8.00	0.43	7.00	0.43	7.00
handball2	0.42	3.00	0.17	22.00	0.38	7.00	0.38	7.00
helicopter	0.37	0.00	0.50	2.00	0.43	2.00	0.43	2.00
iceskater1	0.38	2.00	0.33	5.00	0.37	3.00	0.37	3.00
iceskater2	0.52	3.00	0.34	6.00	0.37	6.00	0.37	6.00
leaves	0.05	6.00	0.17	6.00	0.15	6.00	0.15	6.00
marching	0.73	0.00	0.75	0.00	0.76	0.00	0.76	0.00
matrix	0.60	4.00	0.48	4.00	0.53	5.00	0.53	5.00
motocross1	0.46	1.00	0.34	3.00	0.31	3.00	0.31	3.00
motocross2	0.57	0.00	0.49	1.00	0.38	0.00	0.38	0.00
nature	0.46	4.00	0.27	5.00	0.32	5.00	0.32	5.00
octopus	0.26	1.00	0.25	2.00	0.28	0.00	0.28	0.00
pedestrian1	0.49	9.00	0.48	10.00	0.59	9.00	0.59	9.00
pedestrian2	0.51	1.00	0.34	1.00	0.34	1.00	0.34	1.00
rabbit	0.19	5.00	0.27	4.00	0.24	6.00	0.24	6.00
racing	0.37	0.00	0.35	0.00	0.36	0.00	0.36	0.00
road	0.65	0.00	0.64	0.00	0.65	0.00	0.65	0.00
shaking	0.62	1.00	0.58	8.00	0.51	1.00	0.51	1.00
sheep	0.52	0.00	0.50	0.00	0.51	0.00	0.51	0.00
singer1	0.36	0.00	0.36	0.00	0.36	0.00	0.36	0.00
singer2	0.59	1.00	0.47	3.00	0.55	4.00	0.55	4.00
singer3	0.16	1.00	0.14	1.00	0.15	1.00	0.15	1.00
soccer1	0.34	3.00	0.54	6.00	0.45	2.00	0.45	2.00
soccer2	0.63	10.00	0.07	13.00	0.71	10.00	0.71	10.00
soldier	0.42	1.00	0.24	2.00	0.34	1.00	0.34	1.00

	[TM17]		ab		lab		Grayscale	
Sequence	Ove.	Fail.	Ove.	Fail.	Ove.	Fail.	Ove.	Fail.
sphere	0.68	1.00	0.35	1.00	0.67	1.00	0.67	1.00
tiger	0.67	2.00	0.58	3.00	0.60	3.00	0.60	3.00
traffic	0.68	1.00	0.68	0.00	0.68	0.00	0.68	0.00
tunnel	0.36	0.00	0.51	2.00	0.39	0.00	0.39	0.00
wiper	0.73	1.00	0.73	1.00	0.72	2.00	0.72	2.00
Average	0.48	2.13	0.39	6.58	0.45	3.02	0.45	3.02

TABLE A.1: Performances by varying color space (part 1)

	[TM17]		HSV		Att11		Att50	
Sequence	Ove.	Fail.	Ove.	Fail.	Ove.	Fail.	Ove.	Fail.
bag	0.33	0.00	0.28	0.00	0.26	0.00	0.37	0.00
ball1	0.76	2.00	0.70	6.00	0.73	3.00	0.57	5.00
ball2	0.73	3.00	0.70	3.00	0.00	4.00	0.73	3.00
basketball	0.52	0.00	0.52	3.00	0.57	0.00	0.52	6.00
birds1	0.45	7.00	0.48	3.00	0.45	5.00	0.40	10.00
birds2	0.55	0.00	0.37	2.00	0.37	1.00	0.33	1.00
blanket	0.48	0.00	0.65	1.00	0.46	0.00	0.50	1.00
bmx	0.21	0.00	0.18	0.00	0.21	0.00	0.15	0.00
bolt1	0.48	0.00	0.46	0.00	0.49	0.00	0.49	0.00
bolt2	0.53	0.00	0.50	0.00	0.52	1.00	0.58	1.00
book	0.16	9.00	0.18	8.00	0.19	6.00	0.24	9.00
butterfly	0.44	1.00	0.41	1.00	0.49	2.00	0.40	2.00
car1	0.67	3.00	0.65	2.00	0.54	9.00	0.70	17.00
car2	0.76	0.00	0.79	0.00	0.77	0.00	0.75	0.00
crossing	0.50	1.00	0.50	1.00	0.48	1.00	0.49	1.00
dinosaur	0.41	1.00	0.36	4.00	0.40	4.00	0.36	3.00
fernando	0.37	1.00	0.39	2.00	0.22	2.00	0.38	4.00
fish1	0.35	4.00	0.37	5.00	0.40	4.00	0.36	6.00
fish2	0.22	8.00	0.22	8.00	0.27	6.00	0.23	8.00
fish3	0.44	0.00	0.46	2.00	0.42	1.00	0.44	1.00
fish4	0.28	0.00	0.35	2.00	0.34	2.00	0.34	1.00
girl	0.63	1.00	0.47	2.00	0.55	2.00	0.55	5.00
glove	0.45	3.00	0.43	2.00	0.31	1.00	0.46	3.00

	[TM17]		HSV		Att11		Att50	
Sequence	Ove.	Fail.	Ove.	Fail.	Ove.	Fail.	Ove.	Fail.
godfather	0.51	0.00	0.48	0.00	0.32	2.00	0.44	0.00
graduate	0.34	5.00	0.42	5.00	0.42	6.00	0.38	8.00
gymnastics1	0.42	8.00	0.40	9.00	0.29	7.00	0.46	12.00
gymnastics2	0.58	2.00	0.65	6.00	0.55	5.00	0.67	6.00
gymnastics3	0.31	4.00	0.31	4.00	0.30	4.00	0.31	4.00
gymnastics4	0.42	3.00	0.32	5.00	0.36	3.00	0.26	5.00
hand	0.36	9.00	0.30	9.00	0.41	7.00	0.33	8.00
handball1	0.53	4.00	0.43	7.00	0.39	4.00	0.41	5.00
handball2	0.42	3.00	0.38	7.00	0.37	9.00	0.42	11.00
helicopter	0.37	0.00	0.43	2.00	0.36	0.00	0.37	0.00
iceskater1	0.38	2.00	0.37	3.00	0.28	7.00	0.31	9.00
iceskater2	0.52	3.00	0.37	6.00	0.36	11.00	0.37	11.00
leaves	0.05	6.00	0.15	6.00	0.12	5.00	0.00	7.00
marching	0.73	0.00	0.76	0.00	0.75	0.00	0.75	0.00
matrix	0.60	4.00	0.53	5.00	0.59	3.00	0.57	5.00
motocross1	0.46	1.00	0.31	3.00	0.41	4.00	0.31	3.00
motocross2	0.57	0.00	0.38	0.00	0.23	1.00	0.51	0.00
nature	0.46	4.00	0.32	5.00	0.42	4.00	0.38	5.00
octopus	0.26	1.00	0.28	0.00	0.26	0.00	0.25	0.00
pedestrian1	0.49	9.00	0.59	9.00	0.51	10.00	0.49	10.00
pedestrian2	0.51	1.00	0.34	1.00	0.34	1.00	0.56	1.00
rabbit	0.19	5.00	0.24	6.00	0.41	6.00	0.18	7.00
racing	0.37	0.00	0.36	0.00	0.35	0.00	0.36	0.00
road	0.65	0.00	0.65	0.00	0.62	0.00	0.50	0.00
shaking	0.62	1.00	0.51	1.00	0.53	1.00	0.29	3.00
sheep	0.52	0.00	0.51	0.00	0.50	0.00	0.50	0.00
singer1	0.36	0.00	0.36	0.00	0.36	0.00	0.36	0.00
singer2	0.59	1.00	0.55	4.00	0.62	2.00	0.60	2.00
singer3	0.16	1.00	0.15	1.00	0.14	1.00	0.15	1.00
soccer1	0.34	3.00	0.45	2.00	0.38	3.00	0.43	3.00
soccer2	0.63	10.00	0.71	10.00	0.79	10.00	0.74	11.00
soldier	0.42	1.00	0.34	1.00	0.41	1.00	0.40	2.00
sphere	0.68	1.00	0.67	1.00	0.30	5.00	0.68	1.00
tiger	0.67	2.00	0.60	3.00	0.62	4.00	0.64	2.00

	[TM17]		HSV		Att11		Att50	
Sequence	Ove.	Fail.	Ove.	Fail.	Ove.	Fail.	Ove.	Fail.
traffic	0.68	1.00	0.68	0.00	0.68	0.00	0.69	1.00
tunnel	0.36	0.00	0.39	0.00	0.38	0.00	0.39	0.00
wiper	0.73	1.00	0.72	2.00	0.57	2.00	0.36	2.00
Average	0.48	2.13	0.45	3.02	0.43	3.22	0.44	4.48

TABLE A.2: Performances by varying color space (part 2)

Appendix B

Per sequence results on VOT2015 for different derivatives scales

	[TM17]		$\sigma = 2$		$\sigma = 4$		$\sigma = 8$	
Sequence	Ove.	Fail.	Ove.	Fail.	Ove.	Fail.	Ove.	Fail.
bag	0.33	0.00	0.36	0.00	0.32	0.00	0.32	0.00
ball1	0.76	2.00	0.80	2.00	0.80	1.00	0.68	2.00
ball2	0.73	3.00	0.85	3.00	0.71	3.00	0.03	4.00
basketball	0.52	0.00	0.52	2.00	0.46	3.00	0.48	3.00
birds1	0.45	7.00	0.37	12.00	0.42	9.00	0.47	3.00
birds2	0.55	0.00	0.57	0.00	0.46	0.00	0.55	0.00
blanket	0.48	0.00	0.52	0.00	0.38	0.00	0.43	0.00
bmx	0.21	0.00	0.17	1.00	0.16	1.00	0.15	0.00
bolt1	0.48	0.00	0.54	1.00	0.60	1.00	0.47	2.00
bolt2	0.53	0.00	0.65	0.00	0.64	0.00	0.62	1.00
book	0.16	9.00	0.14	5.00	0.20	8.00	0.13	8.00
butterfly	0.44	1.00	0.52	1.00	0.45	1.00	0.26	1.00
car1	0.67	3.00	0.61	1.00	0.60	0.00	0.72	1.00
car2	0.76	0.00	0.80	0.00	0.72	0.00	0.49	1.00
crossing	0.50	1.00	0.50	1.00	0.49	1.00	0.50	1.00
dinosaur	0.41	1.00	0.34	0.00	0.44	2.00	0.39	2.00
fernando	0.37	1.00	0.43	2.00	0.35	2.00	0.40	2.00
fish1	0.35	4.00	0.36	2.00	0.45	3.00	0.36	4.00
fish2	0.22	8.00	0.25	3.00	0.26	5.00	0.28	6.00
fish3	0.44	0.00	0.54	1.00	0.42	1.00	0.49	0.00
fish4	0.28	0.00	0.42	1.00	0.42	2.00	0.31	1.00
girl	0.63	1.00	0.61	1.00	0.68	1.00	0.62	2.00

	[TM17]		$\sigma = 2$		$\sigma = 4$		$\sigma = 8$	
Sequence	Ove.	Fail.	Ove.	Fail.	Ove.	Fail.	Ove.	Fail.
glove	0.45	3.00	0.45	3.00	0.48	3.00	0.54	3.00
godfather	0.51	0.00	0.40	1.00	0.44	1.00	0.50	2.00
graduate	0.34	5.00	0.39	7.00	0.39	6.00	0.45	12.00
gymnastics1	0.42	8.00	0.34	6.00	0.50	6.00	0.42	5.00
gymnastics2	0.58	2.00	0.62	3.00	0.59	3.00	0.56	5.00
gymnastics3	0.31	4.00	0.28	3.00	0.32	4.00	0.29	3.00
gymnastics4	0.42	3.00	0.33	3.00	0.41	3.00	0.41	0.00
hand	0.36	9.00	0.52	5.00	0.47	8.00	0.33	10.00
handball1	0.53	4.00	0.47	3.00	0.61	2.00	0.47	7.00
handball2	0.42	3.00	0.41	3.00	0.36	2.00	0.50	5.00
helicopter	0.37	0.00	0.42	2.00	0.37	0.00	0.37	0.00
iceskater1	0.38	2.00	0.38	3.00	0.37	4.00	0.32	5.00
iceskater2	0.52	3.00	0.45	3.00	0.47	3.00	0.48	2.00
leaves	0.05	6.00	0.03	6.00	0.13	5.00	0.37	5.00
marching	0.73	0.00	0.74	0.00	0.73	0.00	0.74	0.00
matrix	0.60	4.00	0.50	4.00	0.55	3.00	0.44	3.00
motocross1	0.46	1.00	0.38	3.00	0.24	3.00	0.33	3.00
motocross2	0.57	0.00	0.55	0.00	0.52	0.00	0.31	0.00
nature	0.46	4.00	0.48	4.00	0.48	4.00	0.40	6.00
octopus	0.26	1.00	0.34	1.00	0.31	0.00	0.33	1.00
pedestrian1	0.49	9.00	0.58	7.00	0.64	8.00	0.54	6.00
pedestrian2	0.51	1.00	0.30	0.00	0.31	0.00	0.31	1.00
rabbit	0.19	5.00	0.37	5.00	0.39	4.00	0.27	5.00
racing	0.37	0.00	0.38	0.00	0.42	0.00	0.37	0.00
road	0.65	0.00	0.64	0.00	0.60	0.00	0.68	0.00
shaking	0.62	1.00	0.51	0.00	0.66	1.00	0.57	0.00
sheep	0.52	0.00	0.52	0.00	0.49	1.00	0.53	1.00
singer1	0.36	0.00	0.36	0.00	0.36	0.00	0.34	0.00
singer2	0.59	1.00	0.58	1.00	0.67	1.00	0.64	1.00
singer3	0.16	1.00	0.14	1.00	0.32	1.00	0.33	2.00
soccer1	0.34	3.00	0.48	3.00	0.49	3.00	0.41	2.00
soccer2	0.63	10.00	0.86	13.00	0.88	13.00	0.71	12.00
soldier	0.42	1.00	0.46	1.00	0.43	1.00	0.37	1.00
sphere	0.68	1.00	0.53	1.00	0.56	2.00	0.57	0.00

	[TM17]		$\sigma = 2$		$\sigma = 4$		$\sigma = 8$	
Sequence	Ove.	Fail.	Ove.	Fail.	Ove.	Fail.	Ove.	Fail.
tiger	0.67	2.00	0.69	1.00	0.57	1.00	0.75	1.00
traffic	0.68	1.00	0.68	0.00	0.61	1.00	0.55	1.00
tunnel	0.36	0.00	0.37	0.00	0.37	0.00	0.59	1.00
wiper	0.73	1.00	0.74	1.00	0.70	1.00	0.59	1.00
Average	0.48	2.13	0.48	2.17	0.48	2.21	0.47	2.68

TABLE B.1: Performances by varying derivative scale

Bibliography

[AAR04] Shivani Agarwal, Aatif Awan, and Dan Roth. "Learning to detect objects in images via a sparse, part-based representation". In: *IEEE transactions on pattern analysis and machine intelligence* 26.11 (2004), pp. 1475–1490.

[Ach+10] Radhakrishna Achanta et al. *Slic superpixels*. Tech. rep. 2010.

[Ach+12] Radhakrishna Achanta et al. "SLIC superpixels compared to state-of-the-art superpixel methods". In: *IEEE transactions on pattern analysis and machine intelligence* 34.11 (2012), pp. 2274–2282.

[AOV12] Alexandre Alahi, Raphael Ortiz, and Pierre Vandergheynst. "FREAK: Fast retina keypoint". In: *Computer vision and pattern recognition (CVPR), 2012 IEEE conference on*. Ieee. 2012, pp. 510–517.

[ARS08] Mykhaylo Andriluka, Stefan Roth, and Bernt Schiele. "People-tracking-by-detection and people-detection-by-tracking". In: *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*. IEEE. 2008, pp. 1–8.

[Aru+02] M Sanjeev Arulampalam et al. "A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking". In: *IEEE Transactions on signal processing* 50.2 (2002), pp. 174–188.

[Bal81] Dana H Ballard. "Generalizing the Hough transform to detect arbitrary shapes". In: *Pattern recognition* 13.2 (1981), pp. 111–122.

[Bay+08] Herbert Bay et al. "Speeded-up robust features (SURF)". In: *Computer vision and image understanding* 110.3 (2008), pp. 346–359.

[BCM05] Antoni Buades, Bartomeu Coll, and J-M Morel. "A non-local algorithm for image denoising". In: *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*. Vol. 2. IEEE. 2005, pp. 60–65.

[Ber+16] Luca Bertinetto et al. "Staple: Complementary learners for real-time tracking". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 1401–1409.

[BH01] Kai Briechle and Uwe D Hanebeck. "Template matching using fast normalized cross correlation". In: *Aerospace/Defense Sensing, Simulation, and Controls*. International Society for Optics and Photonics. 2001, pp. 95–102.

[Bha43] A Bhattachayya. "On a measure of divergence between two statistical population defined by their population distributions". In: *Bulletin Calcutta Mathematical Society* 35 (1943), pp. 99–109.

[Bol+10] David S Bolme et al. "Visual object tracking using adaptive correlation filters". In: *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*. IEEE. 2010, pp. 2544–2550.

[BR05] Stanley T Birchfield and Sriram Rangarajan. "Spatiograms versus histograms for region-based tracking". In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. Vol. 2. IEEE. 2005, pp. 1158–1163.

[Bra98] Gary R Bradski. "Computer vision face tracking for use in a perceptual user interface". In: (1998).

[Bre+09] Michael D Breitenstein et al. "Robust tracking-by-detection using a detector confidence particle filter". In: *Computer Vision, 2009 IEEE 12th International Conference on*. IEEE. 2009, pp. 1515–1522.

[Bre01] Leo Breiman. "Random forests". In: *Machine learning* 45.1 (2001), pp. 5–32.

[BVD11] Olivier Barnich and Marc Van Droogenbroeck. "ViBe: A universal background subtraction algorithm for video sequences". In: *IEEE Transactions on Image processing* 20.6 (2011), pp. 1709–1724.

[Béz66] Pierre Bézier. "Définition numérique des courbes et surfaces I". In: *Automatisme* 11.12 (1966), pp. 625–632.

[Can08] Kevin Cannons. "A review of visual tracking". In: *Dept. Comput. Sci. Eng., York Univ., Toronto, Canada, Tech. Rep. CSE-2008-07* (2008).

[CFM10] Raffaele Cappelli, Matteo Ferrara, and Davide Maltoni. "Minutia cylinder-code: A new representation and matching technique for fingerprint recognition". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 32.12 (2010), pp. 2128–2141.

[CG10] Michael Crosier and Lewis D Griffin. "Using basic image features for texture classification". In: *International Journal of Computer Vision* 88.3 (2010), pp. 447–460.

[Cio+15] Andrea Ciolini et al. "Efficient Hough forest object detection for low-power devices". In: *Multimedia & Expo Workshops (ICMEW), 2015 IEEE International Conference on*. IEEE. 2015, pp. 1–6.

[CLL05] Robert T Collins, Yanxi Liu, and Marius Leordeanu. "Online selection of discriminative tracking features". In: *IEEE transactions on pattern analysis and machine intelligence* 27.10 (2005), pp. 1631–1643.

[CM02] Dorin Comaniciu and Peter Meer. "Mean shift: A robust approach toward feature space analysis". In: *IEEE Transactions on pattern analysis and machine intelligence* 24.5 (2002), pp. 603–619.

[CRM03] Dorin Comaniciu, Visvanathan Ramesh, and Peter Meer. "Kernel-based object tracking". In: *IEEE Transactions on pattern analysis and machine intelligence* 25.5 (2003), pp. 564–577.

[Csu+04] Gabriella Csurka et al. "Visual categorization with bags of keypoints". In: *Workshop on statistical learning in computer vision, ECCV*. Vol. 1. 1-22. Prague. 2004, pp. 1–2.

[CV95] Corinna Cortes and Vladimir Vapnik. “Support-vector networks”. In: *Machine learning* 20.3 (1995), pp. 273–297.

[Dan+14a] Martin Danelljan et al. “Accurate scale estimation for robust visual tracking”. In: *British Machine Vision Conference, Nottingham, September 1-5, 2014*. BMVA Press. 2014.

[Dan+14b] Martin Danelljan et al. “Adaptive color attributes for real-time visual tracking”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 1090–1097.

[Dan+16] Martin Danelljan et al. “Beyond correlation filters: Learning continuous convolution operators for visual tracking”. In: *European Conference on Computer Vision*. Springer. 2016, pp. 472–488.

[Den+09] Jia Deng et al. “Imagenet: A large-scale hierarchical image database”. In: *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE. 2009, pp. 248–255.

[Der87] Rachid Deriche. “Using Canny’s criteria to derive a recursively implemented optimal edge detector”. In: *International journal of computer vision* 1.2 (1987), pp. 167–187.

[DG13] Stefan Duffner and Christophe Garcia. “Pixeltrack: a fast adaptive algorithm for tracking non-rigid objects”. In: *Proceedings of the IEEE international conference on computer vision*. 2013, pp. 2480–2487.

[DH72] Richard O Duda and Peter E Hart. “Use of the Hough transformation to detect lines and curves in pictures”. In: *Communications of the ACM* 15.1 (1972), pp. 11–15.

[DT05] Navneet Dalal and Bill Triggs. “Histograms of oriented gradients for human detection”. In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05)*. Vol. 1. IEEE. 2005, pp. 886–893.

[Dub15] Séverine Dubuisson. “Visual Tracking by Particle Filtering”. In: *Tracking with Particle Filter for High-Dimensional Observation and State Spaces* (2015), pp. 1–27.

[Eca+08] Olivier Ecabert et al. "Automatic model-based segmentation of the heart in CT images". In: *IEEE transactions on medical imaging* 27.9 (2008), pp. 1189–1201.

[Eve+07] Mark Everingham et al. "The PASCAL Visual Object Classes challenge 2007 (VOC2007) results". In: (2007).

[Eve+10] Mark Everingham et al. "The PASCAL Visual Object Classes (VOC) challenge". In: *International journal of computer vision* 88.2 (2010), pp. 303–338.

[Fan+10] Jialue Fan et al. "Human tracking using convolutional neural networks". In: *IEEE Transactions on Neural Networks* 21.10 (2010), pp. 1610–1623.

[Fel+10] Pedro F Felzenszwalb et al. "Object detection with discriminatively trained part-based models". In: *IEEE transactions on pattern analysis and machine intelligence* 32.9 (2010), pp. 1627–1645.

[FH75] Keinosuke Fukunaga and Larry Hostetler. "The estimation of the gradient of a density function, with applications in pattern recognition". In: *IEEE Transactions on information theory* 21.1 (1975), pp. 32–40.

[FHX05] Graham D Finlayson, Steven D Hordley, and Ruixia Xu. "Convex programming colour constancy with a diagonal-offset model". In: *IEEE International Conference on Image Processing 2005*. Vol. 3. IEEE. 2005, pp. III–948.

[FLW95] Johan Forsberg, Ulf Larsson, and Ake Wernersson. "Mobile robot navigation using the range-weighted Hough transform". In: *IEEE Robotics & Automation Magazine* 2.1 (1995), pp. 18–26.

[FS95] Yoav Freund and Robert E Schapire. "A decision-theoretic generalization of on-line learning and an application to boosting". In: *European conference on computational learning theory*. Springer. 1995, pp. 23–37.

[Gal+11] Juergen Gall et al. "Hough forests for object detection, tracking, and action recognition". In: *IEEE transactions on pattern analysis and machine intelligence* 33.11 (2011), pp. 2188–2202.

[Gha15] Youness Aliyari Ghassabeh. "A sufficient condition for the convergence of the mean shift algorithm with Gaussian kernel". In: *Journal of Multivariate Analysis* 135 (2015), pp. 1–10.

[Gir+14] Ross Girshick et al. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 580–587.

[GL13] Juergen Gall and Victor Lempitsky. "Class-specific Hough forests for object detection". In: *Decision forests for computer vision and medical image analysis*. Springer, 2013, pp. 143–157.

[Gol+06] S Golemati et al. "Comparison of B-mode, M-mode and Hough transform methods for measurement of arterial diastolic and systolic diameters". In: *Engineering in Medicine and Biology Society, 2005. IEEE-EMBS 2005. 27th Annual International Conference of the*. IEEE. 2006, pp. 1758–1761.

[GRB13] Martin Godec, Peter M Roth, and Horst Bischof. "Hough-based tracking of non-rigid objects". In: *Computer Vision and Image Understanding* 117.10 (2013), pp. 1245–1256.

[Gri06] Lewis D Griffin. "The 2nd order local-image-structure solid". In: *Perception ECVP abstract* 35 (2006), pp. 0–0.

[GRVG12] Juergen Gall, Nima Razavi, and Luc Van Gool. "An introduction to random forests for multi-class object detection". In: *Outdoor and large-scale real-world scene analysis*. Springer, 2012, pp. 243–263.

[Haa10] Alfred Haar. "Zur theorie der orthogonalen funktionensysteme". In: *Mathematische Annalen* 69.3 (1910), pp. 331–371.

[Har09] Peter E Hart. "How the Hough transform was invented [DSP History]". In: *IEEE Signal Processing Magazine* 26.6 (2009), pp. 18–22.

[HAS15] Yang Hua, Karteek Alahari, and Cordelia Schmid. "Online object tracking with proposal selection". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 3092–3100.

[HBS14] Jan Hosang, Rodrigo Benenson, and Bernt Schiele. "How good are detection proposals, really?" In: *arXiv preprint arXiv:1406.6962* (2014).

[Hen+12] João F Henriques et al. "Exploiting the circulant structure of tracking-by-detection with kernels". In: *European conference on computer vision*. Springer. 2012, pp. 702–715.

[Hen+15] João F Henriques et al. "High-speed tracking with kernelized correlation filters". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37.3 (2015), pp. 583–596.

[Ho95] Tin Kam Ho. "Random decision forests". In: *Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on*. Vol. 1. IEEE. 1995, pp. 278–282.

[Hou62] P.V.C. Hough. *Method and means for recognizing complex patterns*. US Patent 3,069,654. Dec. 1962. URL: https://www.google.com/patents/US3069654.

[HS88] Chris Harris and Mike Stephens. "A combined corner and edge detector." In: *Alvey vision conference*. Vol. 15. Citeseer. 1988, p. 50.

[HST11] Sam Hare, Amir Saffari, and Philip HS Torr. "Struck: Structured output tracking with kernels". In: *2011 International Conference on Computer Vision*. IEEE. 2011, pp. 263–270.

[HTS16] David Held, Sebastian Thrun, and Silvio Savarese. "Learning to track at 100 fps with deep regression networks". In: *European Conference on Computer Vision*. Springer. 2016, pp. 749–765.

[IB98] Michael Isard and Andrew Blake. "Condensation: conditional density propagation for visual tracking". In: *International journal of computer vision* 29.1 (1998), pp. 5–28.

[IK88] John Illingworth and Josef Kittler. "A survey of the Hough transform". In: *Computer vision, graphics, and image processing* 44.1 (1988), pp. 87–116.

[IKN+98] Laurent Itti, Christof Koch, Ernst Niebur, et al. "A model of saliency-based visual attention for rapid scene analysis". In: *IEEE Transactions on pattern analysis and machine intelligence* 20.11 (1998), pp. 1254–1259.

[JW+02] Richard Arnold Johnson, Dean W Wichern, et al. *Applied multivariate statistical analysis*. Vol. 5. 8. Prentice hall Upper Saddle River, NJ, 2002.

[KEB91] Nahum Kiryati, Yuval Eldar, and Alfred M Bruckstein. "A Probabilistic Hough transform". In: *Pattern recognition* 24.4 (1991), pp. 303–316.

[Kha+13] Fahad Shahbaz Khan et al. "Coloring action recognition in still images". In: *International journal of computer vision* 105.3 (2013), pp. 205–221.

[KKA00] Nahum Kiryati, Heikki Kälviäinen, and Satu Alaoutinen. "Randomized or Probabilistic Hough transform: unified performance evaluation". In: *Pattern Recognition Letters* 21.13 (2000), pp. 1157–1164.

[KMM10] Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas. "Forward-backward error: Automatic detection of tracking failures". In: *Pattern recognition (ICPR), 2010 20th international conference on*. IEEE. 2010, pp. 2756–2759.

[KMM12] Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas. "Tracking-Learning-Detection". In: *IEEE transactions on pattern analysis and machine intelligence* 34.7 (2012), pp. 1409–1422.

[KPL+] Matej Kristan, Roman Pflugfelder, Ales Leonardis, et al. *The visual object tracking VOT2014 challenge results*.

[Kri+13] Matej Kristan et al. "The visual object tracking VOT2013 challenge results". In: *Proceedings of the IEEE International Conference on Computer Vision Workshops*. 2013, pp. 98–111.

[Kri+15] Matej Kristan et al. "The visual object tracking vot2015 challenge results". In: *Proceedings of the IEEE International Conference on Computer Vision Workshops*. 2015, pp. 1–23.

[Kri+16a] Matej Kristan et al. *A Novel Performance Evaluation Methodology for Single-Target Trackers*. 2016. URL: http://arxiv.org/abs/1503.01313.

[Kri+16b] Matej Kristan et al. *The Visual Object Tracking VOT2016 challenge results*. Springer. 2016. URL: http://www.springer.com/gp/book/9783319488806.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[KVD90] Jan J Koenderink and AJ Van Doorn. "Receptive field families". In: *Biological cybernetics* 63.4 (1990), pp. 291–297.

[LBL15] Pengpeng Liang, Erik Blasch, and Haibin Ling. "Encoding color information for visual tracking: Algorithms and benchmark". In: *IEEE Transactions on Image Processing* 24.12 (2015), pp. 5630–5644.

[Lew95] John P Lewis. "Fast template matching". In: *Vision interface*. Vol. 95. 120123. 1995, pp. 15–19.

[Lin98] Tony Lindeberg. "Feature detection with automatic scale selection". In: *International journal of computer vision* 30.2 (1998), pp. 79–116.

[LK+81] Bruce D Lucas, Takeo Kanade, et al. "An iterative image registration technique with an application to stereo vision." In: *IJCAI*. Vol. 81. 1. 1981, pp. 674–679.

[LLP16] Hanxi Li, Yi Li, and Fatih Porikli. "Deeptrack: Learning discriminative feature representations online for robust visual tracking". In: *IEEE Transactions on Image Processing* 25.4 (2016), pp. 1834–1848.

[LLS08] Bastian Leibe, Aleš Leonardis, and Bernt Schiele. "Robust object detection with interleaved categorization and segmentation". In: *International journal of computer vision* 77.1-3 (2008), pp. 259–289.

[Low99] David G Lowe. "Object recognition from local scale-invariant features". In: *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*. Vol. 2. Ieee. 1999, pp. 1150–1157.

[Man10] Antoine Manzanera. "Local Jet based similarity for NL-Means filtering". In: *Pattern Recognition (ICPR), 2010 20th International Conference on*. IEEE. 2010, pp. 2668–2671.

[Man11] Antoine Manzanera. "Local jet feature space framework for image processing and representation". In: *Signal-Image Technology and Internet-Based Systems (SITIS), 2011 Seventh International Conference on*. IEEE. 2011, pp. 261–268.

[MC15] Priyanka Mukhopadhyay and Bidyut B Chaudhuri. "A survey of Hough Transform". In: *Pattern Recognition* 48.3 (2015), pp. 993–1010.

[Min+04] Florica Mindru et al. "Moment invariants for recognition under changing viewpoint and illumination". In: *Computer Vision and Image Understanding* 94.1 (2004), pp. 3–27.

[MM09] Subhransu Maji and Jitendra Malik. "Object detection using a max-margin hough transform". In: *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE. 2009, pp. 1038–1045.

[MNJ08] Frank Moosmann, Eric Nowak, and Frederic Jurie. "Randomized clustering forests for image classification". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 30.9 (2008), pp. 1632–1646.

[MP13] Mario Edoardo Maresca and Alfredo Petrosino. "MATRIOSKA: A multi-level approach to fast tracking by learning". In: *International Conference on Image Analysis and Processing*. Springer. 2013, pp. 419–428.

[MP14] Mario Edoardo Maresca and Alfredo Petrosino. "Clustering local motion estimates for robust and efficient object tracking". In: *European Conference on Computer Vision*. Springer. 2014, pp. 244–253.

[MSN12] John E McManigle, Richard V Stebbing, and J Alison Noble. "Modified Hough transform for left ventricle myocardium segmentation in 3-D echocardiogram images". In: *Biomedical Imaging (ISBI), 2012 9th IEEE International Symposium on*. IEEE. 2012, pp. 290–293.

[Mur+15] Yusuke Murai et al. "Weighted Hough Forest for object detection". In: *Machine Vision Applications (MVA), 2015 14th IAPR International Conference on*. IEEE. 2015, pp. 122–125.

[MV11] Jiří Matas and Tomáš Vojíř. "Robustifying the flock of trackers". In: *16th Computer Vision Winter Workshop. Citeseer*. Citeseer. 2011, p. 91.

[NBH16] Hyeonseob Nam, Mooyeol Baek, and Bohyung Han. "Modeling and propagating cnns in a tree structure for visual tracking". In: *arXiv preprint arXiv:1608.07242* (2016).

[NH16] Hyeonseob Nam and Bohyung Han. "Learning multi-domain convolutional neural networks for visual tracking". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 4293–4302.

[NKMVG03] Katja Nummiaro, Esther Koller-Meier, and Luc Van Gool. "An adaptive color-based particle filter". In: *Image and vision computing* 21.1 (2003), pp. 99–110.

[Ots75] Nobuyuki Otsu. "A threshold selection method from gray-level histograms". In: *Automatica* 11.285-296 (1975), pp. 23–27.

[PFJ13] Alessandra A Paulino, Jianjiang Feng, and Anil K Jain. "Latent fingerprint matching using descriptor-based hough transform". In: *IEEE Transactions on Information Forensics and Security* 8.1 (2013), pp. 31–45.

[PMB15] Horst Possegger, Thomas Mauthner, and Horst Bischof. "In defense of color-based model-free tracking". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 2113–2120.

[Por05] Fatih Porikli. "Integral histogram: A fast way to extract histograms in cartesian spaces". In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. Vol. 1. IEEE. 2005, pp. 829–836.

[Pér+02] Patrick Pérez et al. "Color-based probabilistic tracking". In: *European Conference on Computer Vision*. Springer. 2002, pp. 661–675.

[Raz+12] Nima Razavi et al. "Latent Hough transform for object detection". In: *European Conference on Computer Vision*. Springer. 2012, pp. 312–325.

[RD06] Edward Rosten and Tom Drummond. "Machine learning for high-speed corner detection". In: *European conference on computer vision*. Springer. 2006, pp. 430–443.

[Red+16] Joseph Redmon et al. "You only look once: Unified, real-time object detection". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 779–788.

[Ren+15] Shaoqing Ren et al. "Faster R-CNN: Towards real-time object detection with region proposal networks". In: *Advances in neural information processing systems*. 2015, pp. 91–99.

[RGVG10] Nima Razavi, Juergen Gall, and Luc Van Gool. "Back-projection revisited: Scalable multi-view object detection and similarity metrics for detections". In: *European Conference on Computer Vision*. Springer. 2010, pp. 620–633.

[RKB04] Carsten Rother, Vladimir Kolmogorov, and Andrew Blake. "GrabCut: Interactive foreground extraction using iterated graph cuts". In: *ACM transactions on graphics (TOG)*. Vol. 23. 3. ACM. 2004, pp. 309–314.

[RM03] Xiaofeng Ren and Jitendra Malik. "Learning a classification model for segmentation". In: *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*. IEEE. 2003, pp. 10–17.

[Ros69] Azriel Rosenfeld. "Picture processing by computer". In: *ACM Computing Surveys (CSUR)* 1.3 (1969), pp. 147–176.

[Roz+16] Denys Rozumnyi et al. "The World of Fast Moving Objects". In: *arXiv preprint arXiv:1611.07889* (2016).

[Rub+11] Ethan Rublee et al. "ORB: An efficient alternative to SIFT or SURF". In: *2011 International conference on computer vision*. IEEE. 2011, pp. 2564–2571.

[Rus+14] Olga Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge". In: *CoRR* abs/1409.0575 (2014). URL: http://arxiv.org/abs/1409.0575.

[SA04] Koichi Sato and Jake K Aggarwal. "Temporal spatio-velocity transform and its application to tracking and interaction". In: *Computer Vision and Image Understanding* 96.2 (2004), pp. 100–128.

[Sat+10] Ravi Kumar Satzoda et al. "Hierarchical additive Hough Transform for lane detection". In: *IEEE Embedded Systems Letters* 2.2 (2010), pp. 23–26.

[SB91] Michael J Swain and Dana H Ballard. "Color indexing". In: *International journal of computer vision* 7.1 (1991), pp. 11–32.

[SBC08] Jamie Shotton, Andrew Blake, and Roberto Cipolla. "Efficiently Combining Contour and Texture Cues for Object Recognition." In: *BMVC*. 2008, pp. 1–10.

[Ser+13] Pierre Sermanet et al. "Overfeat: Integrated recognition, localization and detection using convolutional networks". In: *arXiv preprint arXiv:1312.6229* (2013).

[Shi+94] Jianbo Shi et al. "Good features to track". In: *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on*. IEEE. 1994, pp. 593–600.

[Sme+14] Arnold WM Smeulders et al. "Visual tracking: An experimental survey". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36.7 (2014), pp. 1442–1468.

[Smi78] Alvy Ray Smith. "Color gamut transform pairs". In: *ACM Siggraph Computer Graphics* 12.3 (1978), pp. 12–19.

[SZ14] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[TBA02] Klaus Toennies, Frank Behrens, and Melanie Aurnhammer. "Feasibility of hough-transform-based iris localisation for real-time-application". In: *Pattern Recognition, 2002. Proceedings. 16th International Conference on*. Vol. 2. IEEE. 2002, pp. 1053–1056.

[Tej+14] Alykhan Tejani et al. "Latent-class Hough forests for 3D object detection and pose estimation". In: *European Conference on Computer Vision*. Springer. 2014, pp. 462–477.

[Tia+04] Qi-Chuan Tian et al. "Fast algorithm and application of hough transform in iris segmentation". In: *Machine Learning and Cybernetics, 2004. Proceedings of 2004 International Conference on*. Vol. 7. IEEE. 2004, pp. 3977–3980.

[TM15] Antoine Tran and Antoine Manzanera. "A versatile object tracking algorithm combining Particle Filter and Generalised Hough Transform". In: *Image Processing Theory, Tools and Applications (IPTA), 2015 International Conference on*. IEEE. 2015, pp. 105–110.

[TM16] Antoine Tran and Antoine Manzanera. "Fast growing Hough forest as a stable model for object detection". In: *Image Processing Theory Tools and Applications (IPTA), 2016 6th International Conference on*. IEEE. 2016, pp. 1–6.

[TM17] Antoine Tran and Antoine Manzanera. "Mixing Hough and Color Histogram Models for Accurate Real-Time Object Tracking". In: *International Conference on Computer Analysis of Images and Patterns (to appear)*. 2017.

[TPM08] Oncel Tuzel, Fatih Porikli, and Peter Meer. "Pedestrian detection via classification on Riemannian manifolds". In: *IEEE transactions on pattern analysis and machine intelligence* 30.10 (2008), pp. 1713–1727.

[Tsa97] Du-Ming Tsai. "An improved generalized Hough Transform for the recognition of overlapping objects". In: *Image and Vision computing* 15.12 (1997), pp. 877–888.

[TT+03] Marko Tkalcic, Jurij F Tasic, et al. "Colour spaces: perceptual, historical and applicational background". In: *Eurocon*. 2003.

[VDSGS10] Koen Van De Sande, Theo Gevers, and Cees Snoek. "Evaluating color descriptors for object and scene recognition". In: *IEEE transactions on pattern analysis and machine intelligence* 32.9 (2010), pp. 1582–1596.

[VDW+09] Joost Van De Weijer et al. "Learning color names for real-world applications". In: *IEEE Transactions on Image Processing* 18.7 (2009), pp. 1512–1523.

[VJ01] Paul Viola and Michael Jones. "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*. Vol. 1. IEEE. 2001, pp. I–I.

[VL15] Andrea Vedaldi and Karel Lenc. "Matconvnet: Convolutional Neural Networks for matlab". In: *Proceedings of the 23rd ACM international conference on Multimedia*. ACM. 2015, pp. 689–692.

[vLK16] Luka Čehovin, Aleš Leonardis, and Matej Kristan. *Visual object tracking performance measures revisited*. 2016. URL: http://arxiv.org/abs/1502.05803.

[VM14] Tomáš Vojíř and Jiří Matas. "The enhanced flock of trackers". In: *Registration and Recognition in Images and Videos*. Springer, 2014, pp. 113–136.

[VNM13] Tomas Vojir, Jana Noskova, and Jiri Matas. "Robust scale-adaptive mean-shift for tracking". In: *Scandinavian Conference on Image Analysis*. Springer. 2013, pp. 652–663.

[Wit87] Andrew P Witkin. *Scale-space filtering*. US Patent 4,658,372. Apr. 1987.

[WLY13] Yi Wu, Jongwoo Lim, and Ming-Hsuan Yang. "Online object tracking: A benchmark". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2013, pp. 2411–2418.

[Woh+12] Paul Wohlhart et al. "Discriminative Hough Forests for Object Detection." In: *BMVC*. 2012, pp. 1–11.

[XO09] Lei Xu and Erkki Oja. "Randomized Hough Transform". In: *Encyclopedia of Artificial Intelligence*. IGI Global, 2009, pp. 1343–1350.

[XOK90] Lei Xu, Erkki Oja, and Pekka Kultanen. "A new curve detection method: Randomized Hough Transform (RHT)". In: *Pattern recognition letters* 11.5 (1990), pp. 331–338.

[Yan+11] Hanxuan Yang et al. "Recent advances and trends in visual tracking: A review". In: *Neurocomputing* 74.18 (2011), pp. 3823–3831.

[YGVG10] Angela Yao, Juergen Gall, and Luc Van Gool. "A Hough transform-based voting framework for action recognition". In: *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*. IEEE. 2010, pp. 2061–2068.

[YJS06] Alper Yilmaz, Omar Javed, and Mubarak Shah. "Object tracking: A survey". In: *Acm computing surveys (CSUR)* 38.4 (2006), p. 13.

[ZD14] C Lawrence Zitnick and Piotr Dollár. "Edge boxes: Locating object proposals from edges". In: *European Conference on Computer Vision*. Springer. 2014, pp. 391–405.

[Čeh17] Luka Čehovin. "TraX: The visual Tracking eXchange protocol and library". In: *Neurocomputing* (2017).

ÉCOLE DOCTORALE
INTERFACES
Approches Interdisciplinaires : Fondements, Applications et Innovation

Titre : Représentation d'objets dans des espaces de caractéristiques locales : application à la poursuite de cibles temps-réel et à la détection

Mots clefs : Vision par ordinateur, Poursuite de cibles, Detection d'objets, Espaces de Caracteristiques, Transformée de Hough

Résumé : La représentation visuelle est un problème fondamental en vision par ordinateur. Le but est de réduire l'information au strict nécessaire pour une tâche désirée. Plusieurs types de représentation existent, comme les caractéristiques de couleur (histogrammes, attributs de couleurs...), de forme (dérivées, points d'intérêt...) ou d'autres, comme les bancs de filtres.
Les caractéristiques bas-niveau (locales) sont rapides à calculer. Elles ont un pouvoir de représentation limité, mais leur généricité présente un intérêt pour des systèmes autonomes et multi-tâches, puisque les caractéristiques haut-niveau découlent d'elles.
Le but de cette thèse est de construire puis d'étudier l'impact de représentations fondées seulement sur des caractéristiques locales de bas-niveau (couleurs, dérivées spatiales) pour deux tâches : la poursuite d'objets génériques, nécessitant des caractéristiques robustes aux variations d'aspect de l'objet et du contexte au cours du temps; la détection d'objets, où la représentation doit décrire une classe d'objets en tenant compte des variations intra-classe. Plutôt que de construire des descripteurs d'objets globaux dédiés, nous nous appuyons entièrement sur les caractéristiques locales et sur des mécanismes statistiques flexibles visant à estimer leur distribution (histogrammes) et leurs co-occurrences (Transformée de Hough Généralisée).
La Transformée de Hough Généralisée (THG), créée pour la détection de formes quelconques, consiste à créer une structure de données représentant un objet, une classe... Cette structure, d'abord indexée par l'orientation du gradient, a été étendue à d'autres caractéristiques. Travaillant sur des caractéristiques locales, nous voulons rester proche de la THG originale.
En poursuite d'objets, après avoir présenté nos premiers travaux, combinant la THG avec un filtre particulaire (utilisant un histogramme de couleurs), nous présentons un algorithme plus léger et rapide (100fps), plus précis et robuste. Nous présentons une évaluation qualitative et étudierons l'impact des caractéristiques utilisées (espace de couleur, formulation des dérivées partielles...).
En détection, nous avons utilisé l'algorithme de Gall appelé forêts de Hough. Notre but est de réduire l'espace de caractéristiques utilisé par Gall, en supprimant celles de type HOG, pour ne garder que les dérivées partielles et les caractéristiques de couleur. Pour compenser cette réduction, nous avons amélioré deux étapes de l'entraînement : le support des descripteurs locaux (patchs) est partiellement produit selon une mesure géométrique, et l'entraînement des nœuds se fait en générant une carte de probabilité spécifique prenant en compte les patchs utilisés pour cette étape. Avec l'espace de caractéristiques réduit, le détecteur n'est pas plus précis. Avec les mêmes caractéristiques que Gall, sur une même durée d'entraînement, nos travaux ont permis d'avoir des résultats identiques, mais avec une variance plus faible et donc une meilleure répétabilité.

Title : Object representation in local feature spaces: application to real-time tracking and detection

Keywords : Computer Vision, Object tracking, Object detection, Feature spaces, Hough Transform

Abstract : Visual representation is a fundamental problem in computer vision. The aim is to reduce the information to the strict necessary for a query task. Many types of representation exist, like color features (histograms, color attributes...), shape ones (derivatives, keypoints...) or filterbanks.
Low-level (and local) features are fast to compute. Their power of representation are limited, but their genericity have an interest for autonomous or multi-task systems, as higher level ones derivate from them.
We aim to build, then study impact of low-level and local feature spaces (color and derivatives only) for two tasks: generic object tracking, requiring features robust to object and environment's aspect changes over the time; object detection, for which the representation should describe object class and cope with intra-class variations. Then, rather than using global object descriptors, we use entirely local features and statisticals mecanisms to estimate their distribution (histograms) and their co-occurrences (Generalized Hough Transform).
The Generalized Hough Transform (GHT), created for detection of any shape, consists in building a codebook, originally indexed by gradient orientation, then to diverse features, modeling an object, a class. As we work on local features, we aim to remain close to the original GHT.
In tracking, after presenting preliminary works combining the GHT with a particle filter (using color histograms), we present a lighter and fast (100 fps) tracker, more accurate and robust. We present a qualitative evaluation and study the impact of used features (color space, spatial derivative formulation).
In detection, we used Gall's Hough Forest. We aim to reduce Gall's feature space and discard HOG features, to keep only derivatives and color ones. To compensate the reduction, we enhanced two steps: the support of local descriptors (patches) are partially chosen using a geometrical measure, and node training is done by using a specific probability map based on patches used at this step. With reduced feature space, the detector is less accurate than with Gall's feature space, but for the same training time, our works lead to identical results, but with higher stability and then better repeatability.

Université Paris-Saclay

Espace Technologique / Immeuble Discovery

Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France