TÉCNICO LISBOA

Robust Tracking of Vessels in Oceanographic Airborne Images

Jorge Pedro da Silva Matos

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisors: Prof. Alexandre José Malheiro Bernardino
Prof. Jorge dos Santos Salvador Marques

November 2016

Acknowledgments

It is with great pleasure that I write this section which means the culmination of an important life step.

First and foremost I would like to thank my supervisor, Professor Alexandre Bernardino for the opportunity and guidance throughout this work.

I would like to thank Doctor Ricardo Ribeiro for all the help and insight that made the development of this project possible.

Also, I would like to acknowledge Professor Jorge Marques, my co-supervisor, for teaching and making me interested in computer vision during his course in image processing.

I thank my girlfriend, Adriana de São Vicente, for always being there for me, through the good and bad moments and tolerating the long hours that I had to work during these five years.

I would like to thank my family and my friends, in particular my mother for her support throughout the years. I would like to mention João Silva, Gonçalo Roque, Nuno Lima, Inês Pedro, João Baúto, Miguel Ferreira, Miguel Martinho, Nuno Machado, Hugo Martins, David Marques, and André Mateus that helped me grow as a person and student, and specially for the many good times that we had talking about things other than just our work.

This work was partially supported by project PTDC/EEIPRO/0426/2014.

Abstract

In this work we present and evaluate an algorithm for tracking vessels in oceanographic airborne image sequences. Such sequences are challenging due to sun reflections, wakes, wave crests and fast motions, which significantly degrade the performance of general purpose tracking algorithms. The proposed method is based on state-of-the-art correlation filter tracking complemented with an image segmentation and blob analysis stage. The purpose of this later stage is to re-center the target in the tracking window to compensate for drifts in the correlation filter. We evaluate our proposal using a known benchmark in the field and compare it with general purpose tracking algorithms. Results show that our method beats the general purpose state-of-the-art tracking algorithms in the airborne maritime scenario both in performance and in computation time. The dataset used to perform the evaluations was obtained during the SEAGULL project. We contribute with annotations to this dataset which is available online for further research.

Keywords

Maritime surveillance; tracking; correlation filter; blob; HOG; CNN;

Resumo

Neste trabalho é proposto e avaliado um método para seguimento de embarcações em sequências de imagens oceanográficas de uma perspectiva aérea. Tais sequências são um desafio devido às reflexões solares, rastos das embarcações, ondas e movimentos rápidos, que degradam significativamente o desempenho de algortimos de seguimento. O método proposto é baseado em filtros de correlação, que são o estado de arte em seguimento, complementando com um passo de segmentação de imagens e análise de *blobs*. A finalidade deste passo é recentrar a detecção feita com o filtro de correlação que acumula erro ao longo do tempo. Avaliamos a nossa proposta usando métodos de avaliação de referência nesta àrea e comparamos com outros algortimos de seguimento propostos recentemente. Os resultados mostram que o nosso método bate o estado da arte no cenário marítimo de uma perspectiva aérea, tanto em desempenho e em tempo de computação. As sequências de vídeo usadas para avaliação dos vários métodos de seguimento foram obtidas durante o projecto SEAGULL. Neste trabalho contribuimos com anotações para algumas destas sequências de vídeo que disponibilizamos publicamente na internet.

Palavras Chave

Vigilância marítima; seguimento; filtros de correlação; blob; HOG; CNN;

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

AUC Area Under the Curve

ASEF Average of Synthetic Exact Filters

ASMS Scale-Adaptative Mean-Shift

BB Bounding Box

BCCB Block Circulant with Circulant Blocks

CFT Correlation Filter Tracker

CN Color Names

CNN Convolutional Neural Network

DSST Discriminative Scale Space Tracker

DFT Discrite Fourier Transform

DLT Deep Learning Tracker

FFT Fast Fourier Transform

FHOG Felzenszwalb Histrogram of Oriented Gradients

FPS Frames Per Second

HOG Histogram of Oriented Gradients

KCF Kernelized Correlation Filters

LWIR Long-Wave Infrared

MDNet Multi-Domain Convolutional Neural Network

MEEM Multiple Experts using Entropy Minimization

MOSSE Minimum Output Sum of Squared Error

MUSTer MUlti-Store Tracker

OTB Object Tracking Benchmark

PCA Principal Component Analysis

RBF Radial Basis Function

ROI Region of Interest

SRE Spacial Robustness Evaluation

SRDCF Spatially Regularized Discriminative Correlation Filter

SVM Support Vector Machine

TRE Temporal Robustness Evaluation

UAV Unmanned Aerial Vehicle

VOT Visual Object Tracking

1

Introduction

Contents

1.1 Motivation

Portugal has one of the biggest Exclusive Economic Zones in the world with more than 1.7 million square kilometers. The Portuguese Air Force intervenes in the maritime surveillance of this zone which is subdivided into: a) detection and control of illegal activity, b) illegal immigration, c) detection of maritime pollution, d) control of maritime traffic, e) military operations, f) search and rescue missions and g) control of fishing activities. The most used methods in the maritime surveillance of the Portuguese coastal area are based in coastal radars, vessel-positioning systems, alarm systems, remote sensing and manned aircraft patrols [1].

Recently, there has been a growing interest in the topic of maritime surveillance, specially due to the European migrant crisis of 2015. Recent cases of aviation disasters at sea, which require the location of the aircraft's wreckage, and the case of pirate populated waters on the Somali coast and Malacca Straight [2] [3], are further evidence of the maritime surveillance problem. Frontex is an agency of the European Union that manages the cooperation between national border authorities securing Europe's external borders. The surveillance of large portions of sea usually requires high investments and Frontex struggles with lack of equipment and human resources to perform that job [4].

The SEAGULL project [1] developed an intelligent maritime surveillance system using Unmanned Aerial Vehicles (UAVs) equipped with various types of optical sensors (visible, infrared, multi- and hyper-spectral) - Fig. 1.1. This system is particularly interesting because it is affordable, easy to deploy and with few infrastructure requirements, in contrast to the other systems used today. In this project a fleet of fixed wing UAVs are equipped with computers running vision algorithms for the automatic detection of maritime vessels, whose coordinates are communicated to a coastal ground station via a radio link. The developed algorithms work in real time on the embedded hardware and present a low rate of false detections [5]. Nevertheless, the developed methods still face some challenges such as sun reflections, breaking waves and boat wakes. Multiple video sequences were acquired during this project to create a dataset for research in the field of maritime surveillance. The work presented in this document is an effort to create computer vision algorithms that are robust to this challenging scenarios.

Figure 1.1: UAV used to acquire the SEAGULL project dataset which we use to evaluate our proposed method.

Figure 1.2: Examples of challenging situations from video sequences acquired during the SEAGULL project such as: deployment of smaller vessels, big wakes (with respect to the vessel size), rotations, sun reflections, scale changes and small targets (low resolution).

1.2 Challenges

The main challenges of the method proposed in [5] are the presence of sun reflections, breaking waves and boat wakes. In computer vision some of the main challenges in the field of visual detection and tracking are illumination variation, scale variation, occlusion, deformation, motion blur, fast motion, rotations, out-of-view, background clutter and low resolution as stated by [6]. All of these are present in the airborne maritime scenario but some are more prominent in the setting of this work. Fig. 1.2 shows some examples of challenging situations that are present in the video sequences of the dataset.

Also, many of the recent proposed methods for detection and tracking suffer from high computational complexity which prevents their application in real-time systems.

1.3 Objectives

The focus of our work is the development of a tracking algorithm more robust to the challenging situations mentioned above while keeping the processing requirements low so that it can run in real-time in the embedded hardware on board the UAVs.

1.4 Contributions

The main contributions of this work are as follows. The development of a new approach to maritime vessel tracking, combining an adaptive correlation filter [7] with a local re-detection step consisting of blob analysis for the correction of tracking offsets. The main idea of our method consists in applying

a detection step on the Region of Interest (ROI) to correct the target center when the conditions are favorable, i.e. when the boat is outside of regions with sun glare and boat wakes (low background clutter). These conditions are detected at the blob analysis phase, using a set of heuristics applied to the number, area, and location of segmented blobs. Using this approach, it is possible to maintain the robustness of the correlation filter tracking, allowing it to keep the track during longer time spans. Another contribution is the evaluation of different image features in the maritime scenario and the benchmark of the proposed approach and several state-of-the-art object tracking algorithms, using a dataset of video sequences acquired during the SEAGULL project [1]. The code is available online for validation of the results. The used dataset is composed of thousands of annotated frames which are available online for further research in this area. We also contribute with manually annotated labels to this dataset in the scope of this work. Finally, a paper [8] with some of the work presented in this document was accepted and presented at the OCEANS'16 MTS/IEEE Monterey conference.

1.5 Outline

This work is organized as follows. In Chapter 2 we present the current state-of-the-art in general purpose tracking algorithms and a few existing applications to maritime scenarios. Then, in Chapter 3 we present the methodology behind the proposed approach and its main components: the correlation filter, the image features used, and the blob analysis step. In Chapter 4 we give an overview of the implementation details of our method, the evaluations performed, and the computer system used to run them. In Chapter 5 we describe the used evaluation methods, datasets, and the experiments done to assess the performance of our methods in comparison to the state-of-the-art. Results are presented, illustrating the advantages of our methods both in tracking performance and computation time. Finally, Chapter 6 summarizes the main conclusions of this work and refers to directions for future research.

2

Related work

Contents

Object tracking is one the most important and difficult tasks in computer vision. According to the Object Tracking Benchmark (OTB) [6], several aspects make visual tracking a very challenging task: illumination variation, scale variation, occlusion, deformation, motion blur, fast motion, in-plane rotation, out-of-plane rotation, out-of-view, background clutters, and low resolution.

2.1 Correlation Filter Tracking

Annually, the state-of-the-art tracking algorithms are presented at the Visual Object Tracking (VOT) challenge [9]. In the 2015 competition, some of the top performers were based on correlation filter tracking using different kinds of approaches. Usually, correlation filters are designed to produce correlation peaks in the estimated target location while having low responses in the background. Although effective, the training of these methods was impractical for online tracking until the proposal of the Minimum Output Sum of Squared Error (MOSSE) filter [10]. Later developments extended MOSSE in a number of ways, such as the introduction of kernel methods by the Kernelized Correlation Filters (KCF) [7] tracker, and the Discriminative Scale Space Tracker (DSST) [11], winner of the 2014 VOT challenge, that as the name suggests, proposed a scale estimation method. Both of these methods used multi-channel Histogram of Oriented Gradients (HOG) features [12] instead of the raw image pixels as used by MOSSE. Later, MUlti-Store Tracker (MUSTer) [13] proposed an approach using short-term and long-term tracking methods, working in a complementary manner. In the MUSTer approach, the Correlation Filter Tracker (CFT) works as the short-term tracker while the long-term part is based on key-points and is used to locate the target when the CFT fails.

In the latest VOT competition (2015), the top correlation filter trackers were the Spatially Regularized Discriminative Correlation Filter (SRDCF) [14] and the DeepSRDCF [15], both from the same authors. They introduced a spatial regularization component in the optimization problem to penalize the correlation filter coefficients farther from the target. Recently, features based on Convolutional Neural Networks (CNNs) [15, 16] have shown state-of-the-art results in various visual recognition tasks including visual tracking. The DeepSRDCF introduces the use of CNN multi-channel features to discriminate the target with the VGG-2048 [17] neural network used for image classification. More recently, [16] used the output of multiple layers of a CNN to train multiple correlation filters. The idea behind this approach is that early layers of CNNs have higher spatial resolution allowing for precise localization while the features from deeper layers capture more semantic information and are robust to significant appearance changes.

2.2 Other Tracking Methods

Other tracking methods that are not based in correlation filters also show state-of-the-art results, specially the ones based on deep learning. In [18] the authors propose the Multi-Domain Convolutional Neural Network (MDNet) tracker, winner of the 2015 VOT challenge. In this method, a CNN is pre-trained using a training set of tracking videos. The main aspect of this CNN is that the last layer is reinitialized at the beginning of each new video (domain-specific) while the deeper layers are updated online.

The Deep Learning Tracker (DLT) is proposed in [19]. A stacked denoising autoencoder is trained offline to learn robust generic image features. Online tracking is made using the encoder trained from the previous autoencoder as a feature extractor, and an additional classification layer. One interesting finding is that the filters in the first layer of the trained feature extractor resemble the well known Gabor filters used for edge detection.

Other methods also present good results in many benchmarking criteria and are worth mentioning. The STRUCK [20] method uses a kernelized structured output Support Vector Machine (SVM) with Haar features and histogram features for tracking. The Multiple Experts using Entropy Minimization (MEEM) tracker [21] proposes a multi-expert restoration scheme to address the problem of model drift in online tracking using a linear SVM. The Scale-Adaptative Mean-Shift (ASMS) tracker [22] is based on the popular mean-shift object tracking method. The authors propose various changes to address the problem of scale estimation while processing frames at a high frame rate.

2.3 Detection and Tracking in the Maritime Scenario

Despite all the existing work on visual tracking, very few methods address detection and tracking on airborne images of maritime scenarios. These scenarios present several specific challenges, the most significant ones are sun reflections (illumination variations), waves and wakes originated by the vessel motion (background clutter), and the fast motion of the camera due to UAV maneuvers. Given the top-down perspective of the camera attached to the UAV, other problems exist, such as the in-plane and out-of-plane rotations. Usually, these types of rotations are not present in a motionless camera. The use of infrared cameras has been proposed to reduce the effects of sun reflections, waves and wakes. In [23] the author proposes a method to detect and classify maritime objects in infrared videos recorded from an autonomous platform. A fusion of three detection methods is made to generate hypotheses of possible boat locations in the image. The first is based on a track-before-detect algorithm using spatio-temporal integrated blob strength, the second exploits stable image regions and the third is based on tracking salient points of the image. Next, a two-stage-classification step with SVMs is performed to classify the vessels. In [24] it is proposed an image saliency method and entropy analysis to detect vessels on Long-Wave Infrared (LWIR) images.

When the algorithms have to run on limited computational resources on board the UAV, additional concerns must be taken in their development. A recent algorithm for boat detection in the visible spectrum [5] uses simple blob analysis, based on spatial and temporal constraints and is capable of operation in real-time on board an UAV. In [25] a binary classifier using simple features is used to classify a target as vessel or background from optical satellite images. The classifier has a cascade structure which rejects background clutter in the earlier stages to improve the computational performance. In [26] a complete system for maritime coastal surveillance is proposed. The boat detection is performed using a Haar-like classifier and a temporal filter. The tracking module uses a nearest neighbor policy with the Bhattacharyya distance between the Hue, Saturation, and Value (HSV) histograms, allowing for multiple target tracking.

Upon the analysis of the current state-of-the-art, the main concerns are the computational cost of the approaches that are usually the top contenders in the visual object tracking challenge. Furthermore, from our experience in the application of general purpose tracking methods to maritime scenarios, we have noticed frequent failures in tracking regions with sun reflections, waves and wakes and failure to cope with the rotations present in our setting. On longer sequences the tracking tends to drift, and approaches based on key-points fail due to the typical low resolution and lack of texture of the target.

In this work we propose a method that is able to mitigate many of these problems which we will outline in the next chapter, Chapter 3.

3

Methodology

Contents

3.1 Overall Architecture

The architecture of the proposed system is shown in Fig. 3.1. The main components are a correlation filter module and a blob analysis module. In the tracking problem it is assumed that the initial location of the target (i.e. the first frame) is given as a bounding box. In practice, for a completely autonomous system a detection module is required to initialize the tracking process.

In this work we use a Kernelized Correlation Filters (KCF), following the work of [7]. The correlation filter is initialized by training it with an image patch cropped from the first frame around the target Bounding Box (BB), either using the raw image (grayscale or RGB) or using different kinds of features. In the present work we use raw image pixels, HOG features [12] and CNN features [27]. The area around this patch is then used as the ROI for the detection in the next frame. The computation of the correlations is performed in the frequency domain where convolutions are replaced with element-wise multiplications using the Discrite Fourier Transform (DFT) of the images. The response map, where the maximum corresponds to the estimated target location, is obtained using the inverse DFT. This information is used to update the location of the target in the new image and define a new bounding box around it (BB' in Fig. 3.1). However, due to clutter and noise, the new target estimated position may still be offset with respect to the true position. To improve the tracking accuracy a blob analysis step is performed. The image is segmented in its most representative blobs and if a dominant large blob is present, its centroid and area are used to update the location and scale of the target, thus defining a new bounding box (BB" in Fig. 3.1). Finally, the correlation filter is updated with the information from a new patch around the new detection. In the next sections we will detail the theoretical and practical aspects of the methods used in our implementation.

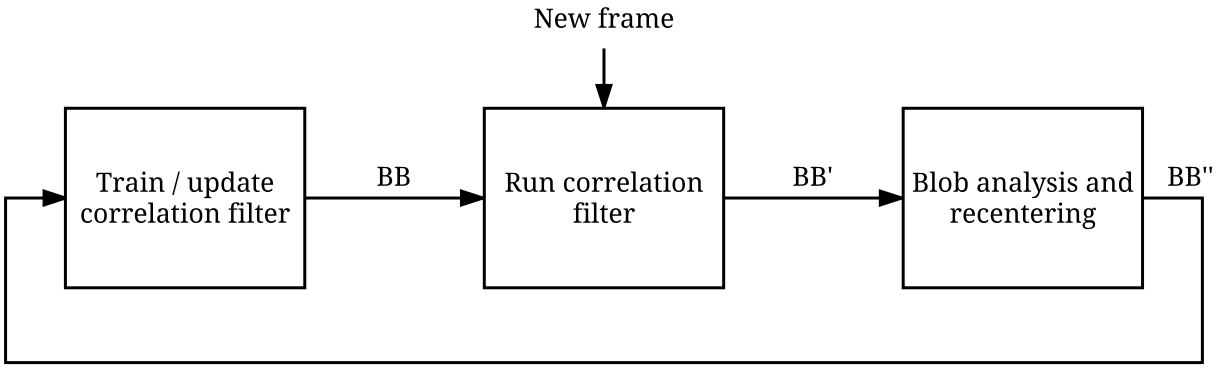


Figure 3.1: System architecture. The system receives a new frame and it runs the correlation filter around the previous bounding box (BB) location returning a updated location of the target (BB'). After, the error of this estimation (BB') is decreased using blob analysis to detect the vessel and recenter the bounding box (BB"). The correlation filter is updated with the information at the new target location.

3.2 Correlation Filters

Correlations filters are specially tuned to a particular image pattern. They are reminiscent to Template Matching techniques in image processing, where a cropped patch around the target h is used to represent the filter. The correlation of this template with the new image x produces a response map y:

$$y = x * h^{-} \tag{3.1}$$

where h^{-} is the reflection in both coordinates of the patch and $*$ is the convolution operator. The location corresponding to the maximum value of the response map will indicate the new position of the target. To improve the computational efficiency of the filter, the convolution is computed in the Fourier domain. Because we are in a discrete and finite domain, the convolution must be replaced by the circular convolution ($\otimes$). This produces some boundary artifacts that can be mitigated by pre-weighting the patches with a windowing function. The computations become:

$$y = x \otimes h^{-} = \mathcal{F}^{-1}(\hat{x} \odot \hat{h}^{*}), \tag{3.2}$$

The hat symbol represents the Discrite Fourier Transform (DFT) of a vector and the $\mathcal{F}^{-1}$ represents the inverse DFT. The $\odot$ represents the element-wise multiplication and superscript $*$ indicates the complex conjugate. For a matter of simplicity, from now on, every-time we refer to convolution we are actually referring to circular convolution.

Using a simple cropped patch to represent the filter produces strong peaks to the target but also responds falsely to the background. To address this problem, methods have been proposed to learn filter coefficients h that produce a more desirable response convolution map y to a given input target x. Typically y is defined as an isotropic Gaussian function with a small standard deviation. A simple way to compute an exact filter is proposed in [28], using the Fourier domain:

$$\hat{h}^{*} = \frac{\hat{y}}{\hat{x}} \tag{3.3}$$

where the division is element-wise. Still, this approach is not very robust to noise and transformations other than pure translations, so [28] proposed the Average of Synthetic Exact Filters (ASEF) algorithm that, as the name suggests, averages multiple exact filters trained for different transformations of the target:

$$\hat{h}^{*} = \sum_{i} \frac{\hat{y_i}}{\hat{x_i}} \tag{3.4}$$

where the x_i are transformed image patches of the target, and y_i are the corresponding desired response maps (Gaussian centered in the location of the target).

One year later, the same author proposed the MOSSE filter that allows better performance than the ASEF with a smaller number of training patterns [10]. The main idea behind MOSSE is that it finds the optimal filter in the least squares sense, h, for multiple sample translations of the target template improving robustness to appearance changes. The MOSSE correlation filter is the solution of:

$$\min_{\hat{h}^*} \sum_i ||\hat{x}_i \odot \hat{h}^* - \hat{y}_i||^2, \tag{3.5}$$

where i indexes each training sample (image). The solution of this optimization problem is given by:

$$\hat{h}^* = \frac{\sum_i \hat{y}_i \odot \hat{x}_i^*}{\sum_i \hat{x}_i \odot \hat{x}_i^*}. \tag{3.6}$$

This method allowed correlation filters to be trained more efficiently and be more robust to variations in lighting, scale, pose and non-rigid deformations.

3.2.1 Kernelized Correlation Filters

Despite the success of the MOSSE filter, it is only composed of linear operations on the input signals. Non-linear geometric or brightness transformations of the target can only be represented by increasing the training set of images, which make the method slower. In [7], it was proposed that the correlation filters can be extended to take advantage of the kernel trick to allow for non-linear operations.

3.2.1.A Linear Case

In [7] the problem is formulated differently than the MOSSE approach. As shown before, the MOSSE filter is the solution of Eq. 3.5, which consist of a filtering operation of the input image, computed in the Discrete Fourier domain. The KCF formulates the problem as a Ridge Regression problem, and by employing the idea of circular shifts and circulant matrices the solution in the linear case is identical to the MOSSE solution. This provides some insight of the MOSSE approach and its implicit operations when computing the correlation of the filter with the target sample in the Fourier domain, which corresponds in evaluating multiple shifts of the base target sample simultaneously. In the following paragraphs we will show the derivations of the KCF.

Let the problem be formulated as a linear regression such that the input x is mapped to the output label as $f(x) = \langle w, x \rangle$, where $\langle \cdot, \cdot \rangle$ means the dot product. Lets assume the one-dimensional case, for a matter of clarity. The 2D case is more difficult to analyze but the properties used for the derivations on the 1D case are also true in the 2D case and the solution is true for both cases as stated by [7]. By deriving first the solution for the 1D case we gain intuition that will help better understand the 2D derivation. Using the Regularized Least Squares (also known as Ridge Regression) the optimization

problem becomes:

$$\min_{w} \sum_{j} (y(j) - f(x_j))^2 + \lambda ||w||^2, \tag{3.7}$$

where x_j is a transformation of the input and $y(j)$ the desired output (scalar), corresponding to the j^{th} entry of y, and λ is a regularization term. Reformulating the problem to use matrix notation we get the following:

$$\min_{w} \|y - Xw\|_2^2 + \lambda \|w\|_2^2 \tag{3.8}$$

where X is a matrix with one sample x_j per row.

The solution to the Ridge Regression problem is known [29] and the closed-form solution is given by:

$$w = (X^H X + \lambda I)^{-1} X^H y, \tag{3.9}$$

where X^H means the Hermitian transpose $X^H = (X^*)^T$ and I is the identity matrix. The Hermitian transpose is used because the computations will be performed in the Fourier domain where values are usually complex.

3.2.1.B Non-Linear Case

In machine learning, it is usual to map the input features (in our case the input images x_j) into a non-linear higher dimensional feature space when the dataset is not linearly separable in the original feature space but might be in a new one (usually we don't know which mapping will make the data linearly separable). The kernel trick is used to avoid having to explicitly map the input into the higher dimensional space.

The kernel trick states that given two vectors v and w in $\mathbb{R}^N$ there are functions that can implicitly compute the dot product between these two vectors in the higher-dimensional space $\mathbb{R}^M$ without explicitly mapping the vectors v and w into $\mathbb{R}^M$. By using the kernel trick, there is no need to work explicitly in the higher-dimensional space because the optimization problem only uses the training examples to compute dot products. In this way we can improve the regression without increasing the computation complexity too much. The increased complexity derives from computing the kernel function which is lower than explicitly mapping the features into the higher-dimensional space.

When using the kernel trick, the input data is mapped implicitly to a non-linear feature space by the function $\varphi(x)$. The kernel function is $\kappa(x, x') = \langle \varphi(x), \varphi(x') \rangle$. The dot products in the linear case are replaced by the kernel function in the non-linear case. Furthermore, the Representer Theorem [30] states that the solution of w from Eq. 3.7 is expressed by a linear combination of the training samples:

$$w = \sum_{i} \alpha_i \varphi(x_i). \tag{3.10}$$

and thus the variables under optimization are now the coefficients α_j.

By incorporating the kernel trick into the problem formulation the regression function becomes:

$$f(x) = \langle w, \varphi(x) \rangle = \sum_j \alpha_j \varphi(x_j) \cdot \varphi(x) = \sum_j \alpha_j \kappa(x, x_j) \tag{3.11}$$

and then the Eq. 3.7 becomes:

$$\min_\alpha \sum_j \left(y(j) - \sum_i \alpha_i \kappa(x_i, x_j) \right)^2 + \lambda \| \sum_i \alpha_i \varphi(x_i) \|^2, \tag{3.12}$$

where the problem is to find the values of α instead of the values of w. Using matrix notation we get:

$$\min_\alpha \| y - K\alpha \|_2^2 + \lambda \alpha^T K \alpha \tag{3.13}$$

and the solution is:

$$\alpha = (K + \lambda I)^{-1} y, \tag{3.14}$$

where K is the kernel matrix with elements $K_{ij} = \kappa(x_i, x_j)$ and I is the identity matrix.

The solutions that we found for the linear case (Eq. (3.9)) and the non-linear case (Eq. (3.14)) could be used in their current form but a large system of linear equations would have to be solved and the computation complexity would not suit a real-time system. In the next sub-section a special case will be presented that will overcome this limitation when all the training samples are composed of cyclical shifts of a base sample.

3.2.1.C Circulant Matrices

Let us assume that all samples are transformations of a base sample x and these transformations are circular shifts. If so, the computation of the inverse matrix in Eq. (3.9) and Eq. (3.14) can be avoided by the introduction of circulant matrices. Let us consider a one-dimensional vector, as we have done for the previous formulations, with a single feature (e.g. pixel value) $x = [x_0, x_1, ..., x_{n-1}]$ where n is the vector length. The solution can then be generalized for 2D images with multiple channels and other multiple channel features as stated in [7] and shown further ahead. This approach allows all the translated samples around the target to be used for training without losing computational efficiency.

One definition of circulant matrix is that the row i of $C(x)$, a circulant matrix with first row x, is given by $P^i x$, where P is the cyclic permutation matrix, a specific permutation matrix that performs a cyclical

shift of x by one element and P^i applies the permutation i times. The permutation matrix is defined as:

$$P = \begin{bmatrix} 0 & 0 & 0 & ... & 1 \\ 1 & 0 & 0 & ... & 0 \\ 0 & 1 & 0 & ... & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & ... & 1 & 0 \end{bmatrix} \tag{3.15}$$

Since the signal x is finite we get the same signal with a period of n shifts ($P^n x = P^0 x$) and all the different signals are defined by:

$$\left\{ P^i x \quad | \quad i = 0, 1, ..., n-1 \right\} \tag{3.16}$$

A $n \times n$ circulant matrix $C(x)$ is obtained from a $n \times 1$ vector x by concatenating all cyclic shifts of the vector x:

$$X = C(x) = \begin{bmatrix} (P^0 x)^T \\ (P^1 x)^T \\ \vdots \\ (P^{n-1} x)^T \end{bmatrix} = \begin{bmatrix} x_0 & x_1 & x_2 & ... & x_{n-1} \\ x_{n-1} & x_0 & x_1 & ... & x_{n-2} \\ x_{n-2} & x_{n-1} & x_0 & ... & x_{n-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_1 & x_2 & x_3 & ... & x_0 \end{bmatrix} \tag{3.17}$$

Circulant matrices have various interesting properties that allow the necessary correlation filter computations to be computationally less expensive. For example, the sum, products and inverses of circulant matrices are also circulant [31]. Also, it is trivial to show that the correlation of two real signals u and v can be computed using the product $C(u)v$ and is equivalent to computing the correlation in the Fourier Domain:

$$C(u)v = \mathcal{F}^{-1}(\hat{u}^* \odot \hat{v}) \tag{3.18}$$

Most interesting of all is the matrix diagonalization property of circulant matrices. The matrix diagonalization process, in the general case, consists of taking a square matrix A and doing the eigen decomposition of this matrix as follows:

$$A = PDP^{-1} \tag{3.19}$$

where P is a matrix composed of the eigenvectors of A, and D is the diagonal matrix whose diagonal elements are the corresponding eigenvalues.

When working with circulant matrices the eigenvectors and eigenvalues have some interesting properties [31]. Every circulant matrix has eigenvectors $y^{(m)} = \frac{1}{\sqrt{n}}(1, e^{-2\pi jm/n}, ..., e^{-2\pi jm(n-1)/n})$ where $m = 0, 1, ..., n-1$ and j is the imaginary unit. The corresponding eigenvalues are:

$$d_m = \sum_{k=0}^{n-1} x_k e^{-2\pi jmk/n}. \tag{3.20}$$

Note that the eigenvalues correspond to the fourier transform of the original signal x. Also, when

composing the matrix P with the eigenvectors of x the result is the DFT matrix F, used to compute the DFT of a vector ($\mathcal{F}(u) = \sqrt{n}Fu$). Replacing this in the Eq. (3.19) we get the following:

$$X = F\text{diag}(\hat{x})F^H \tag{3.21}$$

where F^H is the Hermitian transpose of F.

3.2.1.D Circulant Matrices for the Linear Case

The $X^H X$ term on Eq. (3.9) can be rewritten using Eq. (3.21):

$$X^H X = F\text{diag}(\hat{x}^*)F^H F\text{diag}(\hat{x})F^H \tag{3.22}$$

Since F is a unitary matrix ($F^H F = I$) we obtain:

$$X^H X = F\text{diag}(\hat{x}^*)\text{diag}(\hat{x})F^H \tag{3.23}$$

Operations between diagonal matrices are element-wise and we get the following expression:

$$X^H X = F\text{diag}(\hat{x}^* \odot \hat{x})F^H \tag{3.24}$$

Replacing this on Eq. (3.9) we get:

$$\begin{aligned} w &= \left(F\text{diag}(\hat{x}^* \odot \hat{x})F^H + \lambda I\right)^{-1} X^H y && (3.25)\\ &= \left(F\text{diag}(\hat{x}^* \odot \hat{x})F^H + \lambda F^H I F\right)^{-1} X^H y && (3.26)\\ &= F\text{diag}(\hat{x}^* \odot \hat{x} + \lambda)^{-1} F^H F\text{diag}(\hat{x})F^H y && (3.27)\\ &= F\text{diag}\left(\frac{\hat{x}}{\hat{x}^* \odot \hat{x} + \lambda}\right) F^H y && (3.28) \end{aligned}$$

This is equivalent to

$$Fw = \text{diag}\left(\frac{\hat{x}^*}{\hat{x}^* \odot \hat{x} + \lambda}\right) Fy \tag{3.29}$$

and knowing that $Fz = \hat{z}$, we get

$$\hat{w} = \text{diag}\left(\frac{\hat{x}^*}{\hat{x}^* \odot \hat{x} + \lambda}\right) \hat{y} \tag{3.30}$$

Since the product between a diagonal matrix and a vector is the same as their element-wise product the solution of w can be expressed in the fourier domain as

$$\hat{w} = \frac{\hat{x}^* \odot \hat{y}}{\hat{x}^* \odot \hat{x} + \lambda}, \tag{3.31}$$

where the division is element-wise. Eq. (3.31) allows for better computational efficiency than the solution on Eq. (3.9), being bounded by the DFT operations that have a cost of $\mathcal{O}(n \log n)$, while the Ridge Regression solution has a cost of $\mathcal{O}(n^3)$ bounded by the matrix inversion and products. This is an important property for the system proposed because it allows for real-time computations with great robustness.

What is most interesting is that, even tough we started with a different formulation and made the derivations using a 1D image, the solution obtained is the same as the one obtained with the MOSSE formulation in Eq. (3.6) when using only one sample of the target ($i = 1$). The only difference is the regularization parameter λ that was sometimes used in more recent MOSSE correlation filters to avoid division by zero. To better understand this, lets recall the linear case formulation in Eq. (3.7). For each $i = 0, 1, ..., N - 1$ the following computations are being done:

$$\begin{cases} y(0) = w \cdot x_0 \\ y(1) = w \cdot x_1 \\ ... \\ y(N-1) = w \cdot x_{N-1} \end{cases} \tag{3.32}$$

As we can see from the solutions of MOSSE (Eq. (3.6)) and KCF for the linear case (Eq. (3.9)) $\hat{h}^* = \hat{w}$ and from the properties of the DFT this is equivalent to $h(-n)^* = w(n)$. Since the correlation of two arbitrary signals is defined as

$$u(n) * v^*(-n) = \mathcal{F}^{-1}\left(\mathcal{F}(u(n))\mathcal{F}^*(v(n))\right) \tag{3.33}$$

where $*$ is the convolution operator, we conclude that the signal y is actually the correlation of the signal x with the filter $h(-n)^* = w(n)$,

$$y = x(n) * h(-n)^* = x(n) * w(n) \tag{3.34}$$

and this means that, with the KCF formulation, the objective is, like the MOSSE formulation, to finding the coefficients of the filter $w(n) = h(-n)^*$ that applied to an image x will output the desired response y (correlation). We confirm this in Eq 3.32 because when the signals x_j are circular shifts of the base sample x_0, which is the case, y is the circular correlation of the signal w with x_0.

This solution brings a lot of insight about what is implicitly happening when computing MOSSE correlation filters by formulating the problem as a Ridge Regression problem. Furthermore, the KCF formulation allows the introduction of the non-linear mapping into an higher dimensional space, which was not the case for the MOSSE formulation.

3.2.1.E Circulant Matrices for the Non Linear Case

The previous solution is for the linear case. To use the cyclical shifts method for the non-linear case the kernel matrix K (Eq. (3.14)) has to be circulant. For that to be true one condition has to be imposed – given circulant data $C(x)$, the Kernel matrix K is circulant if the kernel function satisfies $\kappa(x, x') = \kappa(Mx, Mx')$ for any permutation matrix M. This condition arises from Theorem 1 [7,32], which states

Theorem 1. *The matrix K with elements $K_{ij} = \kappa(P^{i-1}x, P^{j-1}x)$ is circulant if κ is a unitarily invariant kernel.*

The proof of Theorem 1 is shown in [7, 32]. Some unitarily kernels are the Gaussian kernel and the linear kernel. First of all, the kernel matrix is $K = C(k^{xx})$ where k^{xx} is the first row of K, x is the training sample and the entries of the kernel matrix are $K_{ij} = \kappa(x_i, x_j)$ for $i, j = 0, 1, ..., N-1$ where N is the size of the training sample as mentioned above. Since the K matrix is circulant if the condition is met we only compute the first row k^{xx}. In [7] $k^{xx'}$ is referred as the kernel correlation of two arbitrary vectors x and x'. Its elements are:

$$k_i^{xx'} = \kappa(x', P^{i-1}x) = \varphi^T(x')\varphi(P^{i-1}x) \tag{3.35}$$

Computing these entries one by one would be computational expensive. Using the previously presented cyclic shifts model overcomes this problem. Radial Basis Function (RBF) kernels meet the condition from Theorem 1. They have the form $\kappa(x, x') = f(||x - x'||^2)$ for some function f. In this work we use the Gaussian kernel, which is a particular case of a RBF. In that case the kernel takes the form $\kappa(x, x') = \exp\left(-\frac{1}{\sigma^2}||x - x'||^2\right)$. The entries of the first row are computed as follows:

$$k_i^{xx'} = \kappa(x', P^{i-1}x) = \exp\left(-\frac{1}{\sigma^2}||x' - P^{i-1}x||^2\right) \tag{3.36}$$

where σ is the standard deviation. Expanding the norm and taking into account that the norm of x does not change independently of the permutation ($||x|| = ||P^{i-1}x||$ for any i) we get:

$$k_i^{xx'} = \kappa(x', P^{i-1}x) = \exp\left(-\frac{1}{\sigma^2}\left(||x||^2 + ||x'||^2 - 2x'^T P^{i-1}x\right)\right) \tag{3.37}$$

Since we can apply this function element-wise to a vector we can replace the term $2x'^T P^{i-1}x$ with $C(x)x'$ to obtain the previous equations in vector form:

$$k^{xx'} = \exp\left(-\frac{1}{\sigma^2}\left(||x||^2 + ||x'||^2 - 2C(x)x'\right)\right) \tag{3.38}$$

and, as previously stated in Eq. (3.18), $C(x)x'$ is the correlation of the real signals x and x' that can be

performed in the Fourier Domain and so the kernel correlation for the Gaussian kernel is computed as:

$$k^{xx'} = \exp\left(-\frac{1}{\sigma^2}\left(||x||^2 + ||x'||^2 - 2\mathcal{F}^{-1}(\hat{x}^* \odot \hat{x}')\right)\right) \tag{3.39}$$

Other useful kernels are the Dot-product kernels. They have the form $\kappa(x, x') = f(\langle x, x' \rangle)$ for a function $f(.)$. The linear case is a special case of the Dot-product kernels where the function $f(.)$ is the identity function $f(\langle x, x' \rangle) = \langle x, x' \rangle$. Each entry of the first row of the kernel matrix K is obtained with:

$$k_i^{xx'} = \kappa(x, P^{i-1}x') = x^T P^{i-1} x' \tag{3.40}$$

As in the case of the Gaussian kernel we can compute the entire vector by applying the function element by element:

$$k^{xx'} = C(x')x = \mathcal{F}^{-1}(\hat{x}^* \odot \hat{x}') \tag{3.41}$$

Computing the entire vector instead of computing the values one by one means the computation complexity is significantly lower. This is the case because the training inputs are shifts of each other and it is possible to take advantage of their representations in the Fourier domain to compute the kernel correlation for all possible shifts only using the DFT and mainly element-wise operations.

Now that we have a way to compute the first line of the kernel matrix K in an efficient way, and knowing that K is circulant, it can be replaced in Eq. (3.14) by $C(k^{xx})$ and using the same properties applied for the linear case it follows that:

$$\begin{aligned}
\alpha &= \left(C(k^{xx}) + \lambda I\right)^{-1} y && (3.42)\\
&= \left(F\mathsf{diag}(\hat{k}^{xx})F^H + \lambda I\right)^{-1} y && (3.43)\\
&= \left(F\mathsf{diag}(\hat{k}^{xx})F^H + \lambda F I F^H\right)^{-1} y && (3.44)\\
&= F\mathsf{diag}(\hat{k}^{xx} + \lambda)^{-1} F^H y && (3.45)
\end{aligned}$$

which is equivalent to multiplying both sides by F^H

$$F^H\alpha = \mathsf{diag}(\hat{k}^{xx} + \lambda)^{-1} F^H y \tag{3.46}$$

and, as before, $F\alpha = \hat{\alpha}$ which is equivalent to $F^H\alpha = \hat{\alpha}^*$

$$\hat{\alpha}^* = \mathsf{diag}\left(\frac{1}{\hat{k}^{xx} + \lambda}\right)\hat{y}^* \tag{3.47}$$

Once more, since the product between a diagonal matrix and a vector is the same as their element-

wise product, the previous equation is equivalent to:

$$\hat{\alpha}^* = \frac{\hat{y}^*}{\hat{k}^{xx} + \lambda} \tag{3.48}$$

Finally, α is obtained by applying the complex conjugate to both members of the equation. Note that the complex conjugate of the division is the division of the complex conjugate and the complex conjugate of a sum is the sum of the complex conjugate. Furthermore λ is a real scalar and k^{xx} is the kernel auto-correlation of the signal x.The auto-correlation of a real function is an even function ($s(t) = s(-t)$). Since the Fourier transform of a real even function is also real the complex conjugate that would appear in the $\hat{k}^{xx}$ is dropped. So, Eq. (3.14) can be written in the Fourier domain by taking advantage of the cyclical nature of the training samples as:

$$\hat{\alpha} = \frac{\hat{y}}{\hat{k}^{xx} + \lambda}, \tag{3.49}$$

where k^{xx} is obtained either using the Gaussian kernel correlation (3.39) or the linear kernel correlation (3.41) using the training sample x.

3.2.1.F 2D Generalization

So far we have shown only the derivations required for the case of a one dimensional signal x, which represents an image with only one row. In practice this is never the case. What is interesting to note is that, as mentioned previously, both the MOSSE and the linear KCF trackers have the same solution even tough the MOSSE formulation always assumed regular image signals (i.e. with 2 dimensions) and KCF assumed images with only one row. Fortunately, the KCF formulation for 1D signals can be generalized for 2D signals. In the next paragraphs we will go through the previous steps but this time for the two dimensional case.

Lets start with the linear case. The notation of the problem formulated in Eq. (3.8) can still be used in the 2D case but with some changes in meaning. In this case X still has one image per row but now we have a vectorization of one image in each row. Lets assume we have the following base sample image x:

$$x = \begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \end{bmatrix} \tag{3.50}$$

then the vectorization of x is:

$$\text{vec}(x) = \begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{21} & x_{22} & x_{23} \end{bmatrix} \tag{3.51}$$

Furthermore, when using cyclical shifts of images, the matrix X is not a circulant matrix but the 2D generalization that is related to the 2D DFT which is a Block Circulant with Circulant Blocks (BCCB) matrix [33, 34]. This is the case because X needs to have all possible horizontal and vertical shifts of

the image. Each shift corresponds to one row of X. Using the base image example x, the matrix X is then:

$$X = \begin{bmatrix} \text{vec}(P^0xP^0) \\ \text{vec}(P^1xP^0) \\ \text{vec}(P^2xP^0) \\ \text{vec}(P^0xP^1) \\ \text{vec}(P^1xP^1) \\ \text{vec}(P^2xP^1) \end{bmatrix} = \left[\begin{array}{ccc|ccc} x_{11} & x_{12} & x_{13} & x_{21} & x_{22} & x_{23} \\ x_{13} & x_{11} & x_{12} & x_{23} & x_{21} & x_{22} \\ x_{12} & x_{13} & x_{11} & x_{22} & x_{23} & x_{21} \\ \hline x_{21} & x_{22} & x_{23} & x_{11} & x_{12} & x_{13} \\ x_{23} & x_{21} & x_{22} & x_{13} & x_{11} & x_{12} \\ x_{22} & x_{23} & x_{21} & x_{12} & x_{13} & x_{11} \end{array}\right] = \begin{bmatrix} B_1 & B_2 \\ B_2 & B_1 \end{bmatrix} \tag{3.52}$$

where $P^i x$ performs i horizontal shifts and xP^j performs j vertical shifts on image x and, B_1 and B_2 matrix are circulant matrices.

Much of the properties of circulant matrices have BCCB matrices equivalents [33,34]. The equivalent property of Eq. (3.21) for BCCB matrices is the following: all BCCB matrices of size $MN \times MN$ are diagonalizable by the unitary matrix $F_M \otimes F_N$, where F_M is the DFT matrix of order M and $\otimes$ is the Kronecker product [33, 34]. If X is a BCCB matrix, then:

$$X = (F_M \otimes F_N)^* D (F_M \otimes F_N) = F_{MN}^* D F_{MN} \tag{3.53}$$

where $F_{MN} = F_M \otimes F_N$ is a unitary matrix ($F_{MN} F_{MN}^H = I$) and D is a diagonal matrix that contains the eigenvalues of X,

$$D = \text{diag}(\text{vec}(\hat{x})) \tag{3.54}$$

where $\hat{x}$ is now the 2D DFT of the 2D signal x, which can be obtain using:

$$\hat{x} = F_M x F_N \tag{3.55}$$

or directly in vector form:

$$\text{vec}(\hat{x}) = (F_M \otimes F_N)\text{vec}(x) \tag{3.56}$$

In the 2D case, w and y are matrices of the same size of x. For generalization purposes x, w and y are of size M by N. In Eq. (3.9) they are replaced by their vectorization, and X is now a BCCB matrix of size MN by MN:

$$\text{vec}(w) = (X^H X + \lambda I)^{-1} X^H \text{vec}(y) \tag{3.57}$$

and as with the 1D case $X^H X$ can be replace by $F_{MN}\text{diag}(\text{vec}(\hat{x}^*)) F_{MN}^* F_{MN} \text{diag}(\text{vec}(\hat{x})) F_{MN}^*$:

$$\text{vec}(w) = (F_{MN}\text{diag}(\text{vec}(\hat{x})^* \odot \text{vec}(\hat{x})) F_{MN}^H + \lambda I)^{-1} X^H \text{vec}(y) \tag{3.58}$$

and by doing the same steps as in the 1D case we arrive at:

$$F_{MN}\text{vec}(w) = \text{diag}(\frac{\text{vec}(\hat{x})^*}{\text{vec}(\hat{x}^*) \odot \text{vec}(\hat{x}) + \lambda})F_{MN}\text{vec}(y) \tag{3.59}$$

and since we know that $F_{MN}\text{vec}(z) = (F_M \otimes F_N)\text{vec}(z) = \text{vec}(\hat{z})$ and the multiplication of a diagonal matrix by a vector is equal to the element-wise multiplication it follows:

$$\text{vec}(\hat{w}) = \frac{\text{vec}(\hat{x})^* \odot \text{vec}(\hat{y})}{\text{vec}(\hat{x})^* \odot \text{vec}(\hat{y}) + \lambda} \tag{3.60}$$

and finally, we can simply undo the vectorization process of each signal since all operations are element-wise. The expression of this solution is the same of the 1D solution.

For the non linear case there is an additional important aspect. To do the same derivations the matrix K has to be a BCCB matrix. In [7], Theorem 2 states:

Theorem 2. *The block matrix K with elements $K_{(ii')(jj')} = \kappa(P^{i-1}xP^{i'-1}, P^{j-1}xP^{j'-1})$ is a BCCB matrix if κ is a unitarily invariant kernel.*

Theorem 2 is a generalization of Theorem 1 for the two dimensional case. In conclusion, K is a BCCB if the kernel function is a unitarily invariant kernel which is the case for the Gaussian kernel and the linear kernel.

K is now obtained from $BCCB(k^{xx})$, where k^{xx} is a matrix of the same size of the image x, with entries $k_{ij} = \kappa(x, P^{i-1}xP^{j-1})$ for $i = 0, 1, ..., M$ and $j = 0, 1, ..., N$. $BCCB(z)$ constructs the BCCB matrix from an arbitrary z matrix. For the Gaussian kernel we now have:

$$k_{ij}^{xx'} = \exp\left(-\frac{1}{\sigma^2}||x' - P^{i-1}xP^{j-1}||_2^2\right) \tag{3.61}$$

$$= \exp\left(-\frac{1}{\sigma^2}(||x'||_2^2 + ||x||_2^2 - 2x'^T P^{i-1}xP^{j-1})\right) \tag{3.62}$$

where $||.||_2$ is the Frobenius norm. Once more, it is possible to compute all the elements with:

$$\text{vec}(k^{xx'}) = \exp\left(-\frac{1}{\sigma^2}(||x'||_2^2 + ||x||_2^2 - 2BCCB(x)\text{vec}(x'))\right) \tag{3.63}$$

where $BCCB(x)vec(x')$ is the 2D correlation of the signals x and x' [33,34] and, changing notation back to matrix notation it can be calculated by $\mathcal{F}^{-1}(\hat{x'}^* \odot \hat{x})$:

$$k_g^{xx'} = \exp\left(-\frac{1}{\sigma^2}(||x'||_2^2 + ||x||_2^2 - 2\mathcal{F}^{-1}(\hat{x'}^* \odot \hat{x}))\right) \tag{3.64}$$

where k_g stands for the Gaussian kernel correlation and, applying the same logic, we obtain the kernel

correlation of the 2D signals x and x' when using a linear kernel:

$$k_{ij}^{xx'} = x^T P^i x' P^j \tag{3.65}$$

$$\text{vec}(k^{xx'}) = BCCB(x)\text{vec}(x') \tag{3.66}$$

$$k_l^{xx'} = \mathcal{F}^{-1}(\hat{x}^* \odot \hat{x}') \tag{3.67}$$

where k_l stands for linear kernel correlation.

Finally, the equivalent to Eq. (3.49) in the 2D case is:

$$\begin{aligned} \text{vec}(\alpha) &= (K + \lambda I)^{-1}\text{vec}(y) && (3.68)\\ &= (F_{MN}^* \text{diag}(\text{vec}(\hat{k}^{xx}))F_{MN} + \lambda I)^{-1}\text{vec}(y) && (3.69)\\ &= F_{MN}\text{diag}(\text{vec}(\hat{k^{xx}}) + \lambda)^{-1}F_{MN}^H\text{vec}(y) && (3.70) \end{aligned}$$

and by multiplying by F_{MN}^H in both members we obtain the equivalent expression:

$$F_{MN}^H\text{vec}(\alpha) = \text{diag}\left(\frac{1}{\text{vec}(\hat{k^{xx}}) + \lambda}\right) F_{MN}^H\text{vec}(y) \tag{3.71}$$

and we know that $F_{MN}^H\text{vec}(z) = \text{vec}(\hat{z}^*)$ for an arbitrary matrix z:

$$\text{vec}(\hat{\alpha}^*) = \frac{\text{vec}(\hat{y}^*)}{\text{vec}(\hat{k}^{xx}) + \lambda} \tag{3.72}$$

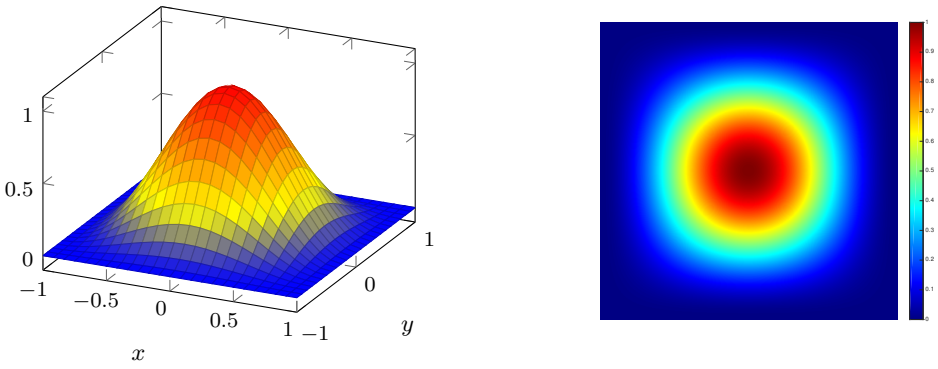
once more, by undoing the vectorization and changing to matrix notation we obtain Eq. (3.49) now generalized for the 2D case:

$$\hat{\alpha} = \frac{\hat{y}}{\hat{k}^{xx} + \lambda} \tag{3.73}$$

3.2.1.G Input Pre-Processing

There is one problem of working with images and the Fourier transform. The images have discontinuities in their borders and due to the periodic nature of the DFT this gives origin to noise in the Fourier representation of the image. In signal processing, to alleviate this problem, it is usual to use a windowing function to weight the signal and smooth the boundary discontinuities.

In this work we use the Hanning window [35,36] to alleviate the boundary discontinuities of the image.



(a) **(b)**

Figure 3.2: Example of a 2D Hanning window used to weight the input image in order to remove the image boundary discontinuities when computing its DFT. (a) 3D plot of the Hanning window. We used N and M equal to 2 to simplify the representation. In practice this values should be the size of the image patch that will be used to train or detect. (b) Color map representation of the Hanning window.

A one dimensional Hanning window of size N is defined as:

$$hann_N(n) = 0.5 - 0.5\cos\left(\frac{2\pi n}{N-1}\right) \quad \text{for} \quad n = 0, 1, ..., N-1 \tag{3.74}$$

and it is possible to obtain a M by N Hanning window (Fig. 3.2) from two one dimensional Hanning windows of size M and N respectively:

$$hann_{M,N} = hann_M \cdot hann_N^T. \tag{3.75}$$

The input image for training or detection has to be weighted using $hann_{M,N}$ before computing any DFT which will result in the border values being weighted to zero. Furthermore, using the weighted window has the secondary effect of resulting in an image that has more focus on the center (center of the target to track) and smoothly less focus in regions that are farther from the target.

3.2.1.H Training and Update Scheme

To actually train the correlation filter we only need to compute the coefficients α. To do that in a computational efficient way (Eq. (3.73)) we require the DFT of the desired output y, that is usually defined as a isotropic 2d Gaussian signal, and the DFT of the kernel auto-correlation k^{xx} of the input image x, weighted with a Hanning window (Eq. (3.75)). k^{xx} can be computed using Eq. (3.64) if the choosen kernel is the Gaussian kernel or Eq. (3.67) if using the linear kernel.

Another important aspect of correlation filters, that was not mentioned so far, is how to update the filter with the new information acquired with the new detections. Usually, the running average is the

(a) **(b)**

Figure 3.3: (a) Grayscale of the original image patch of a vessel to be tracked. (b) Image patch of the target weighted with a Hanning window, Eq. 3.75 and Fig. 3.2(b). Note that the focus of the image is on the target and the boundaries are smoothly converging to zero which attenuates the image boundary discontinuities that affect the DFT.

Algorithm 3.1: Training step of the KCF tracker for the first time instant 0, using the Gaussian kernel.

Input : x, image patch of the target to track;
y, desired output (usually a gaussian signal, with mean at the center of the target);
Output: α_0, coefficients of the correlation filter.

1: Weight x with a Hanning window
2: Compute $\hat{k}_g^{xx} = \mathcal{F}\left(\exp(-\frac{1}{\sigma^2}(||x||^2 + ||x||^2 - 2\mathcal{F}^{-1}(\hat{x}^* \odot \hat{x})))\right)$
3: Compute $\hat{\alpha}_0 = \frac{\hat{y}}{\hat{k}^{xx}+\lambda}$
4: Set $\hat{x}_0 = \hat{x}$

chosen method. In the KCF the coefficients $\hat{\alpha}$ are updated as follows

$$\hat{\alpha}_t = \eta \frac{\hat{y}}{\hat{k}^{xx} + \lambda} + (1 - \eta)\hat{\alpha}_{t-1}, \tag{3.76}$$

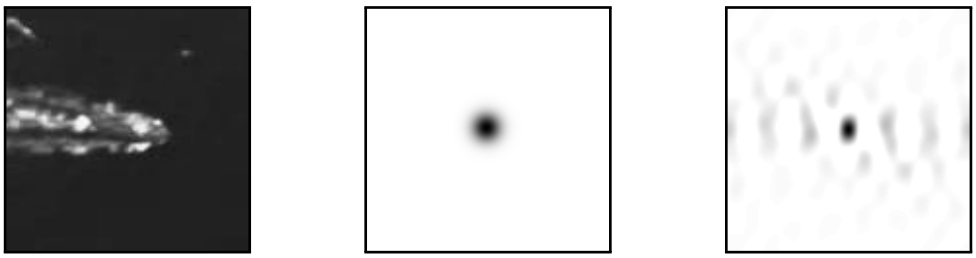
where t defines the current step, and η is the learning rate.

A model of the target is also maintained, which is required for the detection step, as will be shown further ahead. The running average is also the chosen method to update the target model. The model at the first time instant is the image patch of the target selected to track at the first frame. Then, the target model x_t is updated with the image patch of the target at the estimated location in the next instant

$$\hat{x}_t = \eta\hat{x} + (1 - \eta)\hat{x}_{t-1}, \tag{3.77}$$

where x is the new sample extracted from the current estimated position of the target. This is done in the Fourier domain because, as we will see, the detection step requires the DFT of the target model and this way we avoid switching between domains.

The training step algorithm for the first time instant, which we define as 0, is shown in Algorithm 3.1.



(a) (b) (c)

Figure 3.4: (a) Region of interest x, (b) the desired output as a 2D Gaussian signal y and (c) the corresponding coefficients of the correlation filter α of the first frame, assuming the use of raw image pixels as features, obtained using Eq. (3.73) and the Gaussian kernel.

Algorithm 3.2: Training step of the KCF tracker for time instant t, using the Gaussian kernel.

Input : x, new image patch of the target;
x_{t-1}, target model of the previous time instant
y, desired output (usually a Gaussian signal, with mean at the center of the target);
α_{t-1}, coefficients of the correlation filter from the previous time instant $t-1$
Output: α_t, coefficients of the correlation filter for instant t.

1: Weight x with a Hanning window
2: Compute $\hat{k}_g^{xx} = \mathcal{F}\left(\exp(-\frac{1}{\sigma^2}(||x||^2 + ||x||^2 - 2\mathcal{F}^{-1}(\hat{x}^* \odot \hat{x})))\right)$
3: Update $\hat{\alpha}_t = \eta\frac{\hat{y}}{\hat{k}^{xx}+\lambda} + (1-\eta)\hat{\alpha}_{t-1}$
4: Update $\hat{x}_t = \eta\hat{x} + (1-\eta)\hat{x}_{t-1}$

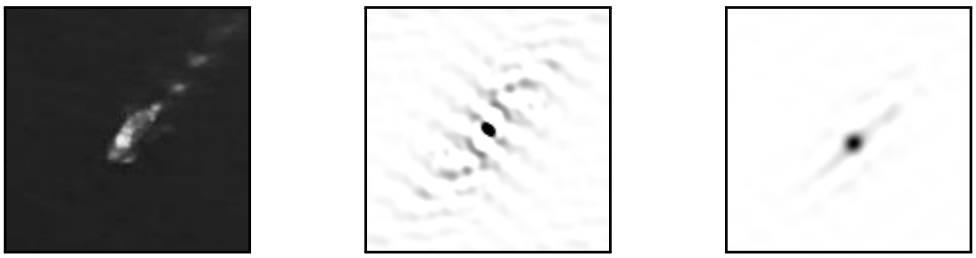
Fig. 3.4 has a practical example of the training step, where we show the input image x, the desired output y and the resulting correlation filter coefficients α. The training step algorithm for an arbitrary time instant t, is shown in Algorithm 3.2.

3.2.1.I Detection

After computing the correlation filter coefficients α, we are ready to make a detection of the target in a new image patch of the same size of the correlation filter. Since it is assumed that the target has a somewhat "continuous" motion (will not jump in the image) we search for the target around its location in the previous frame. Notice that, we are making an estimation of the translation of the target using detection: the KCF approach is referred to as a tracking-by-detection method.

To make the detection we want to compute the regression function $y(m,n) = f(z_{mn})$ where $f(z_{mn}) = \sum_i \sum_j \alpha_{ij}\kappa(z_{mn}, x_{ij})$ (Eq. (3.11)) for all the possible shifts of the test image z, with $m, i = 0, 1, ..., M-1$ and $n, j = 0, 1, ...N-1$. In vector form we can write:

$$vec(y') = K^z vec(\alpha), \tag{3.78}$$



(a) (b) (c)

Figure 3.5: (a) The correlation filter coefficients α trained and updated throughout 226 frames, (b) the candidate patch for detection z after 227 frames since the initialization and (c) the response map y'. Using the trained correlation filter it is possible to estimate the translation of the target relative to the previous position by finding the location of the maximum value of y'.

where K^z is the matrix with entries $K^{x_t z}_{(ij)(mn)} = \kappa(P^{i-1}x_t P^{j-1}, P^{m-1}zP^{n-1})$ and y' is the response map with entries y_{mn}. Once more, using Theorem 2, we know that the matrix K^z is a BCCB matrix obtained with $K^z = BCCB(k^{x_t z})$, where $k^{x_t z}$ is the matrix with entries:

$$k^{x_t z}_{ij} = \kappa(x_t, P^{m-1}zP^{n-1}). \tag{3.79}$$

Recall that we have already shown this equation before for the Gaussian kernel and the linear kernel in Eq. (3.62) and Eq. (3.65) respectively. And, as shown before, we can compute the kernel correlation of the two signals x_t and z with Eq. (3.64) when using the Gaussian kernel and Eq. (3.67) when using the linear kernel.

Given the trained parameter α, the base training samples x_t and the candidate patches for detection z, and knowing how to compute K^z, which is a BCCB matrix, Eq. (3.78) is actually a correlation between two signals $k^{x_t z}$ and α as we have shown before

$$y' = k^{x_t z} \star \alpha \tag{3.80}$$

where $\star$ denotes circular correlation. In the Fourier domain this is equivalent to:

$$y' = \mathcal{F}^{-1}(\hat{k}^{x_t z^*} \odot \hat{\alpha}) \tag{3.81}$$

$$= \mathcal{F}^{-1}(\hat{k}^{x_t z} \odot \hat{\alpha}) \tag{3.82}$$

where the conjugate is dropped because the Fourier transform of the kernel correlation is a real signal, as shown before [7,32]. The translation of the target between frames is estimated by finding the arguments of the maxima of y'.

In Fig. 3.5 it is possible to get a better understanding of these mathematical concepts during the de-

tection step, by analyzing the image patch for detection z, and the color maps of the obtained correlation filter α and respective response map y'. In Algorithm 3.3 we enumerate the steps required to perform the estimation of the target's translation through detection.

Algorithm 3.3: Detection step of the KCF tracker for time instant t, using the Gaussian kernel.

Input : x_t, target model at the current time instant;
z, new image patch containing the target;
α_{t-1}, coefficients of the correlation filter from the previous time instant $t-1$
Output: y', response map

1: Weight z with a Hanning window
2: Compute $\hat{k}_g^{x_t z} = \mathcal{F}\left(\exp(-\frac{1}{\sigma^2}(||x_t||^2 + ||z||^2 - 2\mathcal{F}^{-1}(\hat{x_t}^* \odot \hat{z})))\right)$
3: Compute $y' = \mathcal{F}^{-1}(\hat{k}^{x_t z} \odot \alpha_{t-1})$
4: Determine arguments of the maxima of y'

3.3 Image Features

Correlation filters can be extended to work with any kind of dense features that maintain the spatial information. Examples of these features that have shown good performance are raw image pixels, HOG, Color Names (CN) [37], and more recently CNN features. In this work HOG and CNN features are used. When using raw gray pixels the computations are the ones given in the previous sections. But for features such as RGB, HOG or CNN the kernel correlation has a slight difference because in these cases the input has multiple channels and all derivations were made assuming a single channel.

In [7] it is stated that the integration of multiple channel features does not result in a more difficult inference problem - only some changes to the way the kernel correlation is computed are required. It is proposed that all the channels should be concatenated into a vector. Both kernels used in this work are based on dot-products and norms. The dot-product can be computed by summing the individual dot-products for each channel and from the linearity of the DFT, we can sum the result of each individual channel in the Fourier domain. This means that the equivalent to Eq. (3.67) is, for two arbitrary signals x_a and x_b:

$$k_l^{xx'} = \mathcal{F}^{-1}\left(\sum_c \hat{x}_c^* \odot \hat{x'}_c\right) \tag{3.83}$$

where c identifies the channel number. The equivalent to Eq. (3.64) is:

$$k_g^{xx'} = \exp\left(-\frac{1}{\sigma^2}\left(||x_a||^2 + ||x_b||^2 - 2\mathcal{F}^{-1}\left(\sum_c \hat{x}_{a,c}^* \odot \hat{x}_{b,c}\right)\right)\right). \tag{3.84}$$

3.3.1 Histogram of Oriented Gradient Features

The Histogram of Oriented Gradients (HOG) features used in this work are the ones proposed by Felzenszwalb [12] and are usually called Felzenszwalb Histrogram of Oriented Gradients (FHOG) features. This method builds on the previously proposed HOG [38] that are composed of 36-dimensional features, by stating that essentially the same information can be captured with only 13 dimensions. Furthermore, these lower-dimensional features are extended to include contrast sensitive and contrast insensitive features resulting in 31 dimensions. An overwiew of the method to obtain these features will be given in the following paragraphs.

The first step is to compute pixel-level feature maps. This is done by computing the $\theta(x, y)$ and $r(x, y)$ which are the orientation and magnitude of the gradient at each pixel (x, y). Various methods to compute the gradient of an image can be used but using finite difference filters have shown better results [38]. The finite difference filters are $[-1, 0, 1]$ and $[-1, 0, 1]^T$ to compute the gradient in both directions of the image. In case of a color image only the gradient of the channel that has the largest gradient magnitude is used for each individual pixel.

After computing the gradient for each pixel, the orientation is discretized into p values using either a contrast sensitive (B_1) or contrast insensitive (B_2) definition:

$$B_1(x, y) = \text{round}\left(\frac{p\theta(x, y)}{2\pi}\right) \bmod p \tag{3.85}$$

$$B_2(x, y) = \text{round}\left(\frac{p\theta(x, y)}{\pi}\right) \bmod p. \tag{3.86}$$

To obtain the initial 36-dimensional features, only $B_2(x, y)$ was used, with $p = 9$. This means each bin corresponds to intervals of $\pi/9$ ($20\,\text{deg}$) from 0 to π. In the FHOG features $B_1(x, y)$ are also used with $p = 18$ which means that bins correspond to intervals of $2\pi/18$ ($20\,\text{deg}$) from 0 to 2π. From now on the steps are the same either for B_1 or B_2 and B will be used to denote either in the equations below. The feature map $F(x, y)$ for each pixel (x, y) has the following entries:

$$F(x, y)_b = \begin{cases} r(x, y), & \text{if } b = B(x, y) \\ 0, & \text{otherwise} \end{cases} \tag{3.87}$$

where $r(x, y)$ is the gradient magnitude at pixel (x, y) and $b = 0, 1, ..., p - 1$ which means for each pixel there is a feature vector with p entries. Most entries will be zero except the entry that corresponds to the orientation bin of that pixel, which is the gradient magnitude.

The next step consists in obtaining a cell-based feature map C from the pixel-level feature map F. This is done by defining rectangular "cells" with side length of k pixels. Then each pixel (x, y) is mapped into a cell $(\lfloor x/k \rfloor, \lfloor y/k \rfloor)$ and the feature vector of the cell is the sum (or average) of the pixel-level

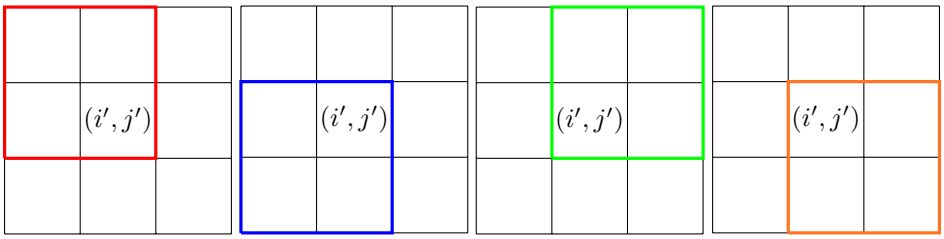


Figure 3.6: Neighborhoods of cell (i', j') used to compute the four different normalization factors. Each color represents one of the four different neighborhoods/ normalization factors.

features mapped into that cell. The cell feature map C, for an image of size $w \times h$ has feature vectors $C(i', j')$ for $0 \leq i' \leq \lfloor (w-1)/k \rfloor$ and $0 \leq j' \leq \lfloor (h-1)/k \rfloor$. Actually, the best results are obtained using a "soft binning" approach where each pixel contributes to the four cells around it using bilinear interpolation [38].

Then, normalization is done to achieve invariance to gain. Multiple normalization approaches exist. Following is the one proposed by [38], where four different normalization factors are used. The factors $N_{m,n}(i', j')$ with $m, n \in \{-1, 1\}$ are computed as

$$N_{m,n}(i', j') = \sqrt{(||C(i', j')||^2 + ||C(i'+m, j')||^2 + ||C(i', j'+n)||^2 + ||C(i'+m, j'+n)||^2)} \tag{3.88}$$

Each factor is interpreted as the "gradient energy" in a square block that contains four cells which include the cell (i', j') and some of its neighbors. Each normalization factor corresponds to a different neighborhood of (i', j') with four blocks (see Fig. 3.6).

The HOG feature map is obtained by concatenating the results of the normalization of the cell-based feature map with each factor and then performing truncation. The element-wise truncation $T_\beta(z)$ of vector z with scalar β consists in replacing the entries of z with the minimum value between that entry and β.

$$H(i', j') = \begin{bmatrix} T_\beta(C(i', j')/N_{-1,-1}(i', j')) \\ T_\beta(C(i', j')/N_{+1,-1}(i', j')) \\ T_\beta(C(i', j')/N_{+1,+1}(i', j')) \\ T_\beta(C(i', j')/N_{-1,+1}(i', j')) \end{bmatrix} \tag{3.89}$$

So far, when using $p = 9$, with 4 normalizations it means that the result has 4×9 = 36-dimensional features for each cells. These are the features proposed by [38].

What [12] later proposed was to do Principal Component Analysis (PCA) on the 36-dimensional feature vectors of a large number of samples from different resolutions. What they found was that linear subspace spanned by the top 11 eigenvectors captures essentially all the information contained in the original 36-dimensional feature vector.

Furthermore, by analysing the structure of the top eigenvectors (refer to [12]), it is possible to see that they are approximately constant along each row or column of their matrix representation (when using

rows to represent the normalizations, and columns to represent the orientation bins, which results in a 4×9 matrix). This way it is possible to define vectors that span a linear subspace that is approximately the subspace spanned by the eigenvectors. Let $V = \{u_1, ..., u_9\} \cup \{v_1, ..., v_4\}$, where the matrix representation of u_k is the matrix with k^{th} column with all entries equal to 1 and 0 otherwise. Eq. 3.90 shows u_2 as an example. The matrix representation of v_k is the matrix with k^{th} row with all entries equal to 1 and 0 otherwise. Eq. 3.91 shows v_3 as an example. The resulting vectors are used to project the 36-dimensional features into a 13-dimensional space. This method is referred to as Analytic Dimensionality Reduction. With this method the feature vector for each cell has 13 dimensions and [12] states that the results show the same performance when using either this method or the PCA with 11 eigenvectors but the Analytic Dimensionality Reduction is computationally less expensive due to the sparse nature of u_k and v_k.

$$u_2 = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.90}$$

$$v_3 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.91}$$

To compute the projection into the subspace the following expression is used:

$$\text{vec}(H'(i', j')) = W \cdot \text{vec}(H(i', j')) = \begin{bmatrix} \text{vec}(u_1)^T \\ \text{vec}(u_2)^T \\ \vdots \\ \text{vec}(u_9)^T \\ \text{vec}(v_1)^T \\ \vdots \\ \text{vec}(v_4)^T \end{bmatrix} \cdot \text{vec}(H(i', j')) \tag{3.92}$$

where W is the matrix constructed with the vectors u_k and v_k in each row. The dimension of W in this case is 13×36.

Doing the same using the contrast sensitive definition in Eq (3.85) with 18 orientation bins, instead of having only 36 dimensions the result has now $4(9 + 18) = 108$ dimensions. 36 from the contrast insensitive definition and 72 from the contrast sensitive definition.

Then, Eq. 3.92 is used to project this 108-dimensional vector into 31-dimensional features. These are composed by the previous 13-features, where 9 are contrast insensitive and 4 capture the overall gradient energy as defined above. The last 18 are obtained from the contrast sensitive definition using the same Analytic Dimensionality Reduction.

Fig. 3.12 shows an example of one channel of the resulting 31 channels after processing a vessel.

3.3.2 Convolutional Neural Network Features

An Artificial neural network, usually referred to only as neural network, is a network that is inspired by the biological neural networks of animals. Neural networks have seen an increase in popularity in the past few years with a multitude of applications, specially in computer vision, such as image classification, image recognition, image segmentation and many others. In essence, neural networks are used to estimate or approximate functions. Below, we will give a simple overview of what constitutes a neural network and how to improve this concept for image processing using convolutional layers. Then, we will explain how to obtain image features from a CNN.

3.3.2.A Perceptron, Sigmoid Neurons and Rectified Linear Unit

A neural network is composed of artificial neurons that connect with other neurons. Two examples of these neurons are the perceptron and the sigmoid neurons.

The perceptron was developed in the 1950s and 1960s by Frank Rosenblatt. It takes multiple binary inputs $x_1, x_2, ..., x_n$ and these inputs are weighted with $w_1, w_2, ..., w_n$. Then, the output (binary) depends if the weighted sum is less than or greater than a defined threshold. In mathematical terms:

$$output = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq threshold \\ 1 & \text{if } \sum_j w_j x_j > threshold \end{cases} \tag{3.93}$$

It is possible to rewrite this definition replacing the sum by the dot-product and moving the threshold to the other side and replacing it with what is usually called bias $b = -threshold$.

$$output = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases} \tag{3.94}$$

An intuitive way to understand the perceptron is that it makes a binary decision by weighting (w_j) evidence x_j. In Fig. 3.7 we can see a depiction of a perceptron.

The main problem of a neural network that uses perceptrons is that small changes to the weights can cause the output to flip (remember the output is binary). This makes it difficult to train the neural network for a desired behaviour.

This problem can be overcome by introducing the sigmoid neuron. These neurons are similar to perceptrons, but are modified to have small changes in the output for small changes in the input. This is done using the sigmoid function $\sigma(z) = 1/(1 + e^{-z})$ (Fig. 3.8(b)). For high values of z this function saturates at 1. For values converging to $-\infty$ the function saturates at 0. The main change from the perceptron is that the discontinuity at the threshold is now smoothed. Actually, the function used in the

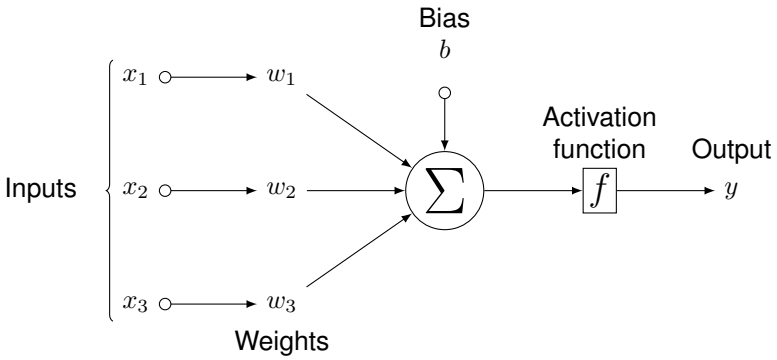
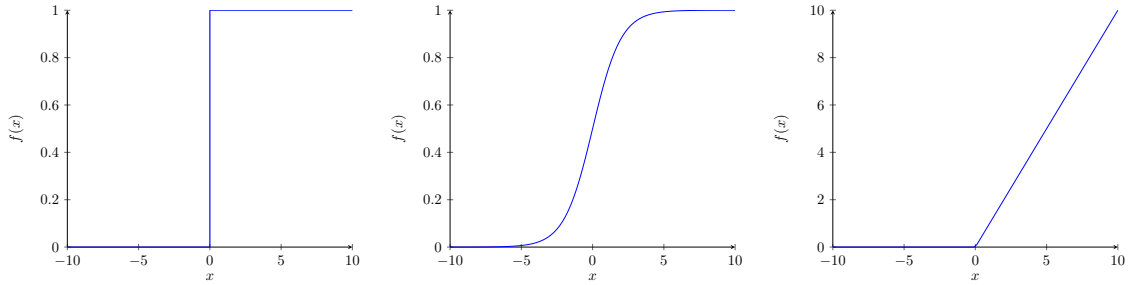


Figure 3.7: Depiction of a neural network neuron. The inputs x_j are binary and $f(.)$ is the step function in the case of a percetron. The inputs are continuous and $f(.) = \sigma(.)$ in case of a sigmoid neuron.



(a) Unit step function **(b)** Sigmoid function **(c)** Rectifier function

Figure 3.8: Activation functions used in different types of artificial neurons.

perceptron is the step function (Fig. 3.8(a)). The output of a sigmoid neuron is:

$$output = \frac{1}{1 + e^{-\sum_j w_j x_j - b}} \tag{3.95}$$

The function that is used in a neuron after computing the weighted sum is called the activation function. Multiple activation functions have been studied in the recent years. In the neural network used in this work the activation function is the rectifier function (Fig. 3.8(c)) and the neuron is called a Rectified Linear Unit (ReLU). This function is defined as

$$ReLU(w \cdot x + b) = \max(0, w \cdot x + b) \tag{3.96}$$

This function is chosen over the sigmoid due to some advantages during training (see vanishing gradient problem [39]) and the ReLU has a more sparse result when compared to the Sigmoid neuron which results in dense representations.

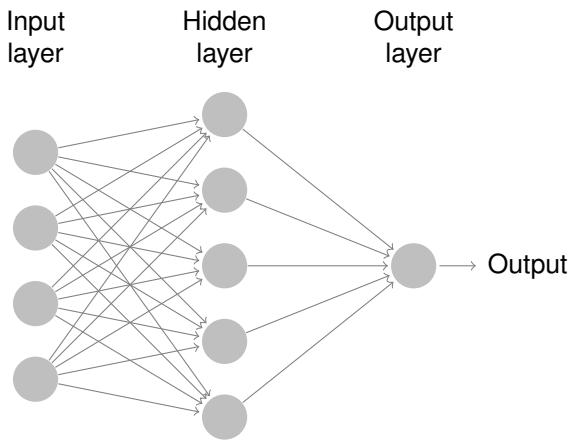


Figure 3.9: Example of a neural network with four inputs, five neurons in the hidden layer and one neuron in the output layer.

3.3.2.B Deep Neural Networks and Convolutional Neural Networks

A neural network is composed of multiple layers, and each layer is composed of multiple neurons. The neurons from one layer serve as the input for the neurons in the next layer. It is easier to understand this concept through the visualization of Fig. 3.9. The neural network can be generalized to have multiple hidden layers, not just one as in our example, and are commonly called deep neural networks. The number of hidden layers and the number of neurons in each layer (both hidden and output), as well as the activation function are characteristics of the neural network. This is what defines the neural network architecture. There are no pre-defined rules to design the architecture of a neural network and researchers use design heuristics to get certain types of behaviour. One example of this is the trade off between the number of hidden layers and the time to train the neural network.

Simple neural networks or deep neural networks are fully connected. This means that every neuron in the network is connected to every neuron in adjacent layers. Images can be inputs for these kinds of networks, with each pixel intensity representing an input x_k. But the downside of doing this is that this network architecture treats input pixels that are far apart the same way it treats pixels that are close together. In other words they do not take into account the spacial structure of the image.

This is where the CNNs come into play. They take into account the spacial structure of the input images and don't require as many weights as the fully connected deep neural networks which in turn allows the training of deeper networks. The three main ideas that distinguish CNNs from regular neural networks are the following.

Local receptive fields. In the fully connected neural networks the input image is usually vectorized and all inputs are connected to each neuron of the second hidden layer. In a CNN the image maintains its structure (matrix) and a region (usually square 3×3, 5×5, etc.) is defined and each hidden neuron only weights the inputs from a region of the defined size. This square slides across the entire image with

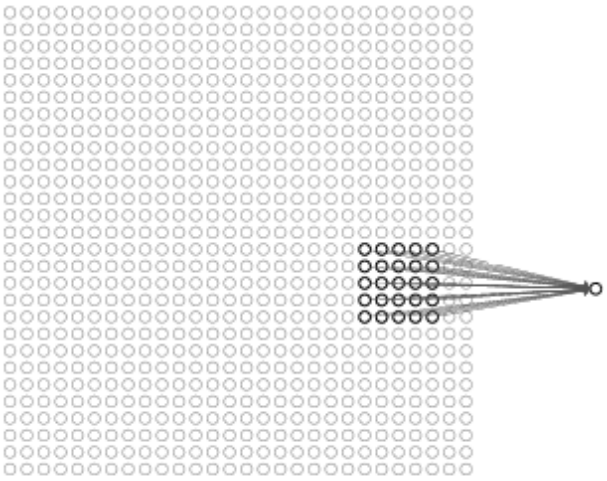


Figure 3.10: Input neurons, in our case an image, and the connections to one of the neurons of the first convolutional layer. Note that only a 5×5 region connects to the one neuron. This is called a receptive field. This square slides around the entire image with a pre-defined stride value.

a stride defined before hand by the network designer. To help visualize this concept see Figure 3.10.

Shared weights and biases. The next main difference from fully connected neural networks is that the connections of the different receptive fields have the same weights. By now it is possible to understand that what is happening is the filtering of the input image and the shared weights define the filter kernel. This is why the CNNs are called convolutional. In other words this means the weights of this convolutional hidden layer are extracting one feature channel, using a learned filter. In practice each hidden layer has multiple filters to detect multiple kinds of features, the same way we extracted 31 channels with the FHOG features.

By sharing weights and biases the number of parameters of a CNN is reduced compared to a fully connected neural network.

Pooling Layers. The last difference of CNNs is that after a convolutional layer it is usual to have a pooling layer. In essence this pooling layer is down-sampling the extracted features with the convolutional layer. The most common used is the max-pooling layer, where a square region is defined and the output of this layer is the maximum values of the multiple square regions. This is done in a sliding window fashion. See Figure 3.11 for an example of a CNN with a simple architecture.

3.3.2.C VGG Convolutional Network

As usual in supervised learning, CNN's are trained from a labeled training set in order to produce a desired output. One example is the classification of images such as the images from ImageNet [40]. The VGG-Net [27] won the first and second place in the ImageNet ILSVRC-2014 localization and classification tasks respectively. In the VGG-Net, a preprocessed image is given as input and this image will

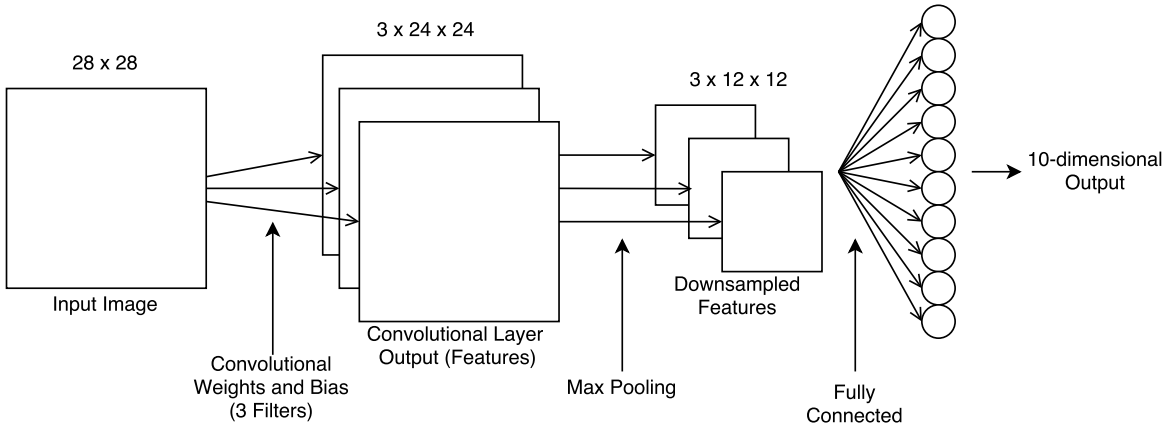


Figure 3.11: Example of a CNN architecture with image input of size 28 by 28, followed by one convolutional layer with 3 filters each with size 5 by 5. This results in 3 feature maps of size 24 by 24 (due to the size of the filter). Then a max pooling layer of 2 by 2 downsamples the features resulting in 3 feature maps of size 12 by 12. The last layer is a fully connected layer outputing a 10-dimensional vector. We can see the convolutional layer as feature extraction step and the last fully connected layer as a classifiers with 10 classes.

Table 3.1: Representation of the VGG-net with 19 layers. The convolutional layers have filters of size 3 by 3. The number of feature maps of each layer is the last number. FC-N means fully connected with N neurons. The layers with bold text are the ones used in this work.

Input	conv3-64 conv3-64	maxpool	conv3-128 conv3-128	maxpool	conv3-256 conv3-256 conv3-256 **conv3-256**	maxpool	conv3-512 conv3-512 conv3-512 **conv3-512**	maxpool	conv3-512 conv3-512 conv3-512 **conv3-512**	maxpool	FC-4096 FC-4096 FC-1000	sof-max

go through 19 layers that are composed of convolutional layers with the rectification (ReLu) non-linearity, max-pooling and fully connected layers. The last layer gives the confidence of an image being one of the 1000 different classes. The appearance of the images is encoded in the neural network weights and the activations of the convolutional hidden layers can be used as multiple channel features for the correlation filter. To extract these features we feed the image to the CNN and use the activation of selected convolutional hidden layers as features. As stated above, the VGG-net used in this work is composed of 19 layers and has hundreds of filters in each layer. Visually representing such a network would result in a figure hard to interpret. The representation of the VGG-net used is shown in Table 3.1.

Note that different layers or even different convolutional neural networks can be used to extract features and the results can vary with the choice. We evaluate the use of the 8th, 12th and 16th layer (not counting maxpooling layers) of the VGG-Net [27] with 19 layers that outputs 256 two-dimensional channels for the 8th layer and 512 two-dimentional channels for the 12th and 16th layers (see Table 3.1 or original paper architecture E [27]). In Fig. 3.12, one of the 256 used channels of the extracted features with the 8th layer can be examined.

Remember that, as is the case when using raw image pixels to train the correlation filter, each feature

(a) (b) (c)

Figure 3.12: (a) Original image, (b) channel 28 of the FHOG features from a total of 31 and one channel from the CNN features extracted from the 8th layer of the VGG-net, from a total of 256 channels.

channel has to be multiplied by a Hanning window to alleviate the boundary effects when computing the DFT. This is true for both the FHOG and CNN features. When using FHOG, [7] states that the Gaussian kernel has better results, as we confirmed experimentally. When testing the CNN features, our results show that the linear kernel achieves a better performance than the Gaussian kernel. Consequently, when using CNN features, we changed the algorithms to use the linear kernels k_l^{xx} and $k_l^{x_t z}$.

3.4 Blob Analysis

Besides having been successfully applied to different problems in computer vision such as object detection based on properties such as brightness or color, the results in [5] show interesting properties of a blob analysis framework in the setting of this work: tracking boats in maritime images. Because the appearance of a boat on a maritime background has a blob-like appearance, we propose the use of image segmentation and blob analysis to detect the maritime vessel in the region of interest (instead of the whole image) and to correct the correlation filter tracker with this detection. By using an image segmentation method and finding the contours of the vessel it is possible to adjust the bounding box to the target, solving to some extend the rotation and scale problem.

3.4.1 Image Segmentation and Morphologic Operations

Before the detection of the connected components, usually referred to as blobs, the image patch of interest has to be segmented. What this means it that we want to separate the image into multiple parts or segments (sets of pixels) that share certain characteristics. In this setting the images are usually composed of two main components: i) vessels and ii) the ocean. At first sight this might seem an easy task but there are a lot of challenging situations that make this problem more complex such as illumination changes, background clutter, high frequency noise, which in the maritime setting translate to highly irregular waves and wakes and regions of intense sun reflections.

Multiple methods to perform image segmentation have been developed through the years due to the vast set of possible applications. Some are based on clustering methods, others on edge detection or texture analysis. What was proposed in [5] was a threshold method. The threshold method proposed is based on two conditions that require two threshold values to be defined before hand. It also assumes that the vessels have bright pixels with high RGB values. After, some filtering is done to remove false detections. This thresholding method works well in that setting because the goal is to detect vessels in the entire frame, which presents a different set of difficulties.

In our case, we want to make a detection around a smaller region, compared to the entire image, around the CFT estimated location of the target. In this case the majority of the image patches should be composed of the two main components referred above (vessels and the ocean). Thus we adopt a binarization approach via the Otsu's method [41] to separate bright and dark parts of the image that, in principle, will correspond to the boats and the ocean surface, respectively. This method is specially pertinent for images that contain two main components. Following we will go into detail about this thresholding method.

3.4.1.A Otsu's Binarization Method

The Otsu's method [41], named after its author, finds a threshold that separates the two peaks of the image histogram from a bimodal grayscale image. Bimodal means that it has two main components, which is the case in our setting. In mathematical terms, the algorithm exhaustively searches for the threshold t that minimizes the intra-class variance (the variance within the class), defined as a weighted sum of the variances of the two classes:

$$\sigma_{\omega}^2(t) = \omega_0(t)\sigma_0^2(t) + \omega_1(t)\sigma_1^2(t), \tag{3.97}$$

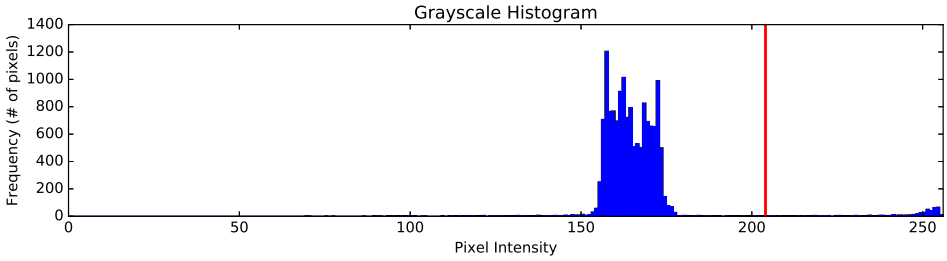
where $\omega_{0,1}$ are the class probabilities separated by the threshold t and $\sigma_{0,1}^2(t)$ are the class variances. The classes probabilities are computed from the histogram that has the count of the number of pixels of each grayscale intensity from 0 to 255 (Fig. 3.13(c)),

$$w_0(t) = \sum_{i=0}^{t-1} \frac{\text{\# of pixels with intensity } i}{\text{\# of pixels}} = \sum_{i=0}^{t-1} p(i) \tag{3.98}$$

$$w_1(t) = \sum_{i=t}^{255} \frac{\text{\# of pixels with intensity } i}{\text{\# of pixels}} = \sum_{i=t}^{255} p(i) \tag{3.99}$$

It is possible to show that minimizing the intra-class variance is the same as maximizing what is called the inter-class variance (the variance between the class), i.e. the threshold with the maximum inter-class variance is the threshold with the minimum intra-class variance. First, the class mean and

(a) (b)



(c)

Figure 3.13: (a) Example of a vessel. (b) Output of the Otsu's image binarization method when the input is image (a). (c) Grayscale histogram of the image patch (a). The threshold value obtained was 204 and is represented in the histogram as the red vertical line.

the mean can be computed as follows:

$$\mu_0(t) = \sum_{i=0}^{t-1} i\frac{p(i)}{\omega_0} \tag{3.100}$$

$$\mu_1(t) = \sum_{i=t}^{255} i\frac{p(i)}{\omega_1} \tag{3.101}$$

$$\mu = \sum_{i=0}^{255} ip(i). \tag{3.102}$$

Knowing the following relations,

$$\omega_0\mu_0 + \omega_1\mu_1 = \mu \tag{3.103}$$

$$\omega_0 + \omega_1 = 1 \tag{3.104}$$

we can derive the expression for inter-class variance

$$\sigma_b^2(t) = \sigma^2 - \sigma_\omega^2(t) \tag{3.105}$$

$$= \omega_0(\mu_0 - \mu)^2 + \omega_1(\mu_1 - \mu)^2 \tag{3.106}$$

$$= \omega_0(t)\omega_1(t)[\mu_0(t) - \mu_1(t)]^2 \tag{3.107}$$

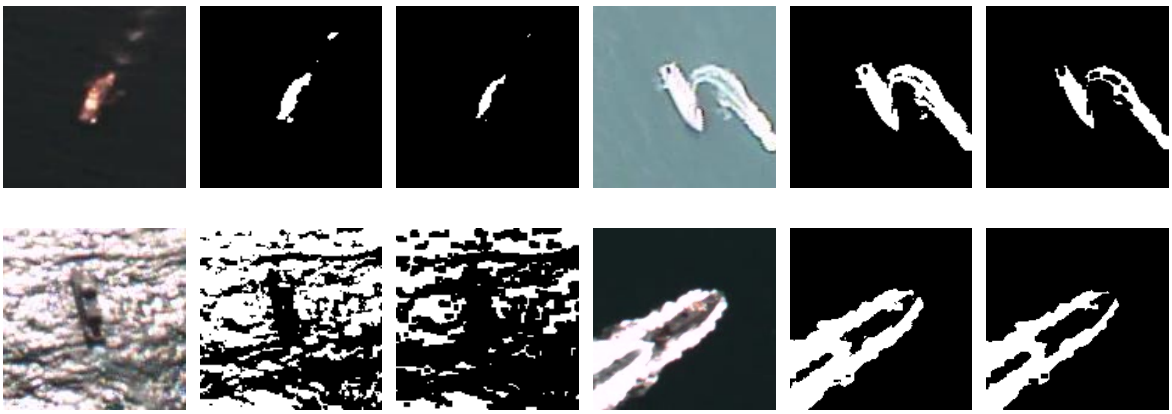


Figure 3.14: Original image, segmented image and eroded image (left to right). Two successful examples (top) and two failure cases (bottom). The failure cases are consequence of the presence of sun-reflections (left) and a big sized wake compared with the vessel size (right).

Finding the maximum inter-class variance is the chosen method since it is faster to compute compared to the inter-class variance. This way the class probabilities and class means can be computed iteratively.

As stated above the result of the Otsu's method which we call the segmented image is a binary image (pixel values 0 or 1). After the Otsu's method the segmented image is eroded (Fig. 3.14) to isolate the vessel from nearby distractors like wakes, life rafts or other vessels, and also to remove some of the noise originated from waves and sun reflections.

3.4.1.B High Frequency Noise and Image Segmentation

Notice in Fig. 3.14 that the Otsu's binarization method failed to perform the segmentation of the target when the vessel is going through a region with intense sun reflections and when the wake is bigger than the vessel. This was to be expected because, in these situations, our image patch is not composed of only two components. Furthermore, in the first case, the pixel intensity is changing with high frequency and this results in the segmented image showing a greater number of blobs, with different sizes, all around the image.

In the presence of highly irregular wakes or waves, as can be seen in Fig. 3.14, this can result in a segmented image with many blobs that do not represent the target and can interfere in its detection. The erosion step, a morphological operation that, as the name indicates, erodes the connected components, removes some of these blobs, but is not enough to overcome this problem.

Following, we will show a method that allows us to overcome the problems mentioned above, and will allow the integration of the blob analysis step into the KCF framework and take advantage of the strength of both methods.

3.4.1.C Context Analysis

After analyzing a set of problematic situations where the image segmentation has poor results we propose the use of context analysis (set of heuristics) to determine if the vessel is going through a region that is challenging for the image segmentation method.

The segmented images of vessels going through regions with sun-reflections usually have the two following characteristics:

1. The number of blobs present in the segmented image is higher than the general case;

2. There are blobs touching the border of the segmented image.

Also, when the vessels are traveling at high speeds, the wakes are bigger than the general case and are also highly irregular. In this case the second statement is also true. The last problematic situation is the appearance of small blobs that might belong to regions of small waves, wakes or sun-reflections.

With all the previous information in mind we defined the following heuristic rules to determine if the conditions near the vessel are favorable for image segmentation:

1. Are there blobs touching the border?

2. Is the number of blobs higher than a threshold T_n?

3. Does the candidate blob have an area smaller than a threshold T_a?

The two first conditions filter the presence of sun-reflections. The first condition filters the presence of highly irregular waves originated from vessels traveling at high speeds. The last one filters false detections of small waves, wakes and reflections.

If none of these conditions are met, we correct the track to the center of the chosen blob (we will show how to choose the blob in the next section). Otherwise we accept the location estimated by the correlation filter without correcting its position.

3.4.2 Detection

So far we already have a segmented image of the area around the location of the target estimated by the correlation filter. Assuming the error of the correlation filter is not bigger than our search area, the target is present in this segmented image. We already filtered the situations that are challenging to segment with the Otsu's method. The next step is to detect the blobs location in the image, finding their contours, center of mass and area. The next section outlines the methods used to achieve this.

3.4.2.A Contour Detection and Image Moments

In [42], the author proposed a method that finds the sequence of coordinates of the borders of connected components in a binary image. With this method we can obtain the contours of the blobs of the segmented image as a sequence of coordinates for each blob.

Once we have the contours of the different blobs we can use the image moments to determine important characteristics of each blob. An image moment is a particular weighted average of the image pixels intensities. They are identical to the moments used in mechanics and statistics. In mathematical terms the $p+q$ order moment for an image with pixel intensities $I(x,y)$ is calculated by:

$$M_{ij} = \sum_x \sum_y x^i y^j I(x,y). \tag{3.108}$$

In our case we want to compute the moments for each blob, which so far are represented by their contours. To compute their individual moments the Green's theorem is used. Green's theorem [43] relates a line integral around a closed plane curve C to a double integral over the plane region D bounded by C. Let C be a positively oriented, piecewise smooth, simple closed curve in a plane, and let D be the region bounded by C. If $M(x,y)$ and $N(x,y)$ are continuous and have continuous first-order partial derivaties in D, then

$$\oint_c (Mdx + Ndy) = \iint_D \frac{\partial M}{\partial x} - \frac{\partial N}{\partial y} dxdy \tag{3.109}$$

where $\oint_c$ denotes a counterclockwise line integral along C.

In practice we want the equivalent to Green's theorem in the discrete domain to compute the contour moments. Multiple versions of the discrete Green's theorem exist as stated in [44]. In the same work a method is proposed to compute the exact moments using a discrete Green's Theorem.

Some definitions of moments are particular pertinent since they compute characteristics of the contours (or image) such as center of mass, area, perimeter, and image orientation. The area of a contour is obtained from the zero order moment:

$$area = M_{00} = \sum_x \sum_y I(x,y) \tag{3.110}$$

It is easy to understand that this expression corresponds to summing the number of pixels in the blob. We use this value to validate the condition defined in the previous section which requires a minimum area to accept a detection.

The center of mass is obtained using the following moments:

$$c_x = \frac{M_{10}}{M_{00}} \tag{3.111}$$

$$c_y = \frac{M_{01}}{M_{00}} \tag{3.112}$$

After computing the centers of each blob we have a set of centers $C^i = (c_x^i, c_y^i)$ for $i = 0, ..., n-1$ where n is the number of blobs in the segmented image. What we propose is to correct the track to the blob that is closest to the CFT estimated location of the target. This is called nearest neighbor approach and corresponds to solving a nearest neighbor search problem:

$$\min_i ||p - C^i||^2 \tag{3.113}$$

where p is the location estimated by the correlation filter.

One of the main challenges of current tracking methods is how to adjust the bounding box to the target that has rotated or suffered appearance changes. Using the contour of the target we can adjust a up-right rectangular bounding box to it. In practice, since we eroded the segmented image, we compensate this by increasing the size of the obtained bounding box by 25%. This has the added effect of including more context around the target which increases the performance of the correlation filter. The complete steps of the proposed method are presented in Algorithm 3.4.

In this chapter we presented the math behind the KCF tracker and our proposed blob analysis step. We also showed how to integrate both steps in order for the algorithm to have the advantages of both methods: robust tracking during sun reflections with the KCF step and robust tracking with rotations and appearance changes with the blob analysis step. The latter also tackles the problem of long-term tracking. In Chapter 4 we will go into detail about the implementation of the proposed method as well as the evaluation methodology.

Algorithm 3.4: Complete steps of the integration of blob analysis into KCF.

Input : x_t, target model at the current time instant;
α, coefficients of the correlation filter;
Output: (c_x, c_y), new location of the target;
(w, h), width and height of the bounding box adjusted to the target

if first frame **then**
  Train correlation filter
else
  Estimate translation with the correlation filter
  Perform binary image segmentation around the location of the previous estimation
  if no blobs touch the border **and** the number of blobs is smaller then the threshold T_n **then**
    Determine the blob that is closest to the correlation filter estimated location
    if the blob's area is higher than a threshold T_a **then**
      Set the track to the center of that blob
      Set new width and height to 1.25 the size of the bounding box adjusted to the blob contour.
    end if
  end if
  Update the correlation filter
end if

4

Implementation

Contents

4.1 Workstation

The proposed method is composed of many complex operations, some of which are computationally expensive either in processing time or memory requirements. Furthermore, we evaluate the proposed method against other state-of-the-art tracking methods which also have high computer specification requirements. The most notable requirement to run our method is a graphics card with 2GB of VRAM when using the CNN features. If this requirement is met it is possible to run our evaluation method with all the proposed tracking methods. We show the computer specifications of the workstation used to run the experiments in Table 4.1.

Table 4.1: Computer specifications of the system used to run and evaluate our method against other state-of-the-art tracking methods in the maritime setting. Version of the software used to implement our method (Python) and evaluate other tracking methods (Matlab) is also shown.

Processor	Intel Xeon CPU W3503 @ 2.40GHz
Memory RAM	12 GB
Graphics Card	GeForce GTX 750
VRAM	2048MB
Operating System	Ubuntu 14.04
Python	2.7
Matlab	R2015a
CUDA	7.0

4.2 Method

4.2.1 Kernelized Correlation Filter Implementation

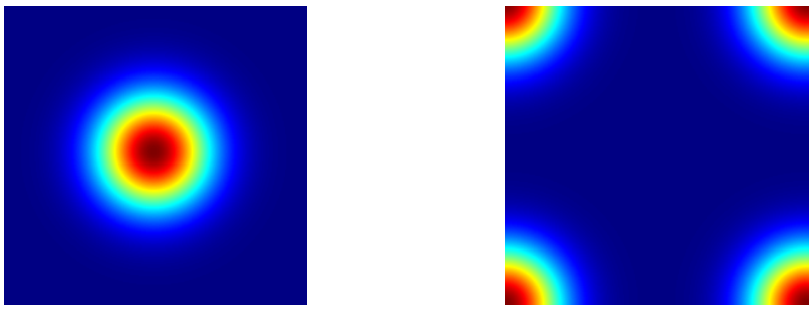
The implementation of the proposed method was made in Python and can be accessed online[1]. To evaluate this method, the parameters used for the correlation filter and blob analysis were as follows. The desired output y for the training step is a M by N two-dimensional Gaussian signal with its peak at the target center:

$$y(m,n) = e^{-\frac{(m-M/2)^2+(n-N/2)^2}{2\sigma^2}}, \tag{4.1}$$

where $m = 0, 1, 2, ..., M$, $n = 0, 1, 2, ..., N$, and $\sigma = 0.1 \cdot \sqrt{a}$ is the peak width, proportional to the target's bounding box area a. The region of interest is defined as a square 2.5 times larger than the target bounding box, which defines the M and N values. The regularization weight λ used was 10^{-4}.

Note that the response map obtained with Eq. (3.82) (regressions for all cyclical shifts) has a subtle detail. Even tough we defined the peak value to the center of the target, which corresponds to the

[1] https://github.com/Magnesiam/CFT_OTB/



(a) (b)

Figure 4.1: Both ways to define the desired output i.e. the regression targets.

base sample with no cyclical shifts, the regression target of the centered sample is actually the top-left element (entry y_{11}) Fig. 4.1. The explanation comes from the fact that after computing the cross-correlation between images in the Fourier domain and converting back to the spacial domain, it is the top-left corner that corresponds to a shift of zero [7, 45]. Either way, due to the cyclical nature of the problem the Gaussian peak actually wraps around the corners as can be seen in Fig. 4.1. It is common to place the Gaussian peak in the middle of the regression target but this requires the correlation output to be shifted by half a window in both directions during detection. Otherwise we could shift the desired output by half a window which would result in a correlation output that does not require these shifts and saves in computation time.

The last detail is that the required Fourier transform for this problem is the unitary DFT. This way the L2-norm of the images is preserved. Most implementations of the Fast Fourier Transform (FFT) do not compute the unitary DFT. In that case the result of the FFT has to be divided by $\sqrt{MN}$ for a signal of size $M \times N$. The inverse unitary DFT has to be multiplied by the same constant factor.

4.2.2 Image Features Extraction

The compute the FHOG features the implementation proposed in the Dlib [46] library was used. The cell size used to compute these features was $k = 4$.

To extract the CNN features we used the VGG-net [27] with 19 layers (architecture E). The model, trained using the ImageNet ILSVRC-2012 dataset is available online in the Caffe Zoo [47], ready to be used with the Caffe framework. The input image has to be reshaped to size 224×224, and the VGG-net image input should have its channels in BGR instead of RGB. The mean pixel value of the ImageNet ILSVRC-2012 dataset has to be subtracted to each pixel of the input image. The features are then extracted from the output of the 8th, 12th and 16th hidden layers. This output has to be upsampled to the size of the correlation filter.

Table 4.2: Parameters used in the KCF tracker implementation using different features. Here a is the area of the initial bounding box. Cell size translates into how much the features were downsampled in regards to the original image.

KCF parameters	Raw	FHOG	CNN
Kernel function	Gaussian	Gaussian	Linear
Gaussian kernel bandwith σ	0.2	0.5	NA
Learning rate η	0.075	0.02	0.02
Spatial bandwidth for desired output		$\sqrt{a}/10$	
Regularization factor λ		10^{-4}	
Cell size	1	4	4

Some parameters of the KCF tracker depend on the type of features that we are using. We show the used values in Table 4.2. These values are the ones recommended by [7] and [16].

4.2.3 Blob Analysis Implementation

For the image segmentation method we use the OpenCV Library [48] implementation of the Otsu's method [41]. We also use this library to find the contours of blobs as proposed by Suzuki [42] and to compute the image moments for each detected blob.

The threshold used for the number of blobs was $T_n = 2$ and the minimum area threshold was $T_a = 80$ pixels. These are the threshold required for the context analysis step.

We use a linear nearest neighbor approach to choose the blob to correct the track. This consists in computing the distance of all detected blobs to the correlation filter estimated location of the target and finding which blob has the minimum distance.

With the contours of the chosen blob we adjust a rectangular bounding box to the target. Then we increase it by 25% to compensate for the erosion step and give some context around the target for the correlation filter. In practice we filter the bounding box size. What this means is that we saturate the change in bounding box size to a maximum of 25% compared to the previous to avoid adjusting it to noisy / false detections.

4.3 Evaluation

To evaluate our method we use the approach proposed by the Object Tracking Benchmark (OTB) [6]. The OTB is a framework that allows the benchmark evaluation of online visual tracking algorithms. This framework has code that implements standard evaluation metrics and proposes new evaluation methods that evaluate robustness to different types of initializations. It also provides a dataset with one hundred video sequences from recent literature with ground truth annotations. We will go into further details

on the evaluation metrics and evaluation methods of the OTB framework in Chapter 5. Regarding the dataset, it contains video sequences from a lot of different settings. Since our objective is to evaluate our tracking method in sequences of airborne maritime images we disregard this dataset and replace it with our own. More details about our dataset can be consulted in Chapter 5.

In this chapter we outlined the implementations details of our proposed methods and the system specifications in which the algorithm was tested. In Chapter 5 we will go into further details about the evaluation method, the used dataset and enumerate the benchmarked tracking methods. To conclude we will present the obtained results.

5

Experiments

Contents

5.1 Evaluation Methods

As stated in Chap. 4, we use the Object Tracking Benchmark (OTB) framework to evaluate the method proposed in this work and to benchmark it with other state-of-the-art general purpose tracking algorithms. This methodology evaluates the tracking methods by computing Precision and Success plots of the tracking under two different initialization strategies denoted Temporal Robustness Evaluation (TRE) and Spacial Robustness Evaluation (SRE). Following we will give details about this framework regarding the evaluation metrics used, Precision and Success, and the evaluation methods TRE and SRE.

5.1.1 Evaluation Metrics

One of the metrics used by the OTB [6] framework is Precision. This metric is based in center location error, defined as the Euclidean distance between the center locations of the tracked target and the manually labeled ground truth. In the past it was usual to average over the center location error and use that result to summarize the performance of the tracker. This had the downside of not discriminating when the target was lost. In this case the tracker's output location is usually random and affects the average result. Recently, the most usual approach is to determine the percentage of frames whose estimated location error is below a given threshold [32, 49]. This percentage is computed for an interval of threshold values: 0 to 50 pixels with a step of 1 pixel. The score chosen to rank the trackers is percentage value for a threshold of 20 pixels as suggested by [6].

The other used metric is Success. The Success plot evaluates the bounding box overlap of the detection with the ground truth. Mathematically, given a detected bounding box r_d and the ground truth bounding box r_{gt} the overlap ratio is given by:

$$S = \frac{|r_d \cap r_{gt}|}{|r_d \cup r_{gt}|}, \tag{5.1}$$

where $\cap$ and $\cup$ represent the intersection and union, respectively, and $|.|$ represents the number of pixels in that region. The Success plot shows the percentage of frames whose bounding box overlap ratio is higher than a given threshold for threshold values from 0 to 1, with steps of 0.05, where 1 means perfect match of the detection and ground truth and 0 meaning lost target. To rank the different algorithms, the Area Under the Curve (AUC) of each Success plot is used.

The most usual way to evaluate tracking methods is to run them through multiple test sequences, initializing with the ground truth bounding box of the first frame. The downside of this approach is that the trackers might be sensitive to the initialization. To take this into account in the evaluation, the OTB framework adds spatial (SRE) and temporal (TRE) perturbations to the initialization.

5.1.2 Temporal Robustness Evaluation and Spacial Robustness Evaluation

The Temporal Robustness Evaluation (TRE) consists in initializing the trackers at different frames, not just the first, and running them until the end of the sequence. Each sequence is evaluated by initializing in 20 different frames. The initial frames are chosen by starting with the first frame of the sequence and stepping through them at a regular interval. The step is approximately the number of frames of the sequence divided by 20.

The Spacial Robustness Evaluation (SRE) consists in introducing error in the initialization by shifting the bounding box by 10% of the target size in 8 different directions, and scaling it by 0.8, 0.9, 1.1 and 1.2 of the ground truth size. This results in 12 different initializations.

Both the TRE and SRE are pertinent because in a real world scenario the trackers are usually initialized with a vessel detector that is likely to introduce error in the initialization. This way we can evaluate their robustness in that regard.

5.2 Dataset

The OTB framework has available a dataset recommended to evaluate visual tracking methods. As previously mentioned, we are more interested on evaluating the tracking algorithms in airborne maritime video sequences. Because of this, we disregard the OTB dataset and use the one acquired by the SEAGULL project [1, 5, 24].

5.2.1 Dataset Acquisition

During the SEAGULL project, multiple video sequences were acquired with multiple types of optical sensors (visible, infrared, multi- and hyper- spectral) on board an UAV (Fig. 1.1). In this work we will focus on evaluating the tracking algorithms on video sequences from the visible and the LWIR spectrum.

The visible spectrum dataset used for the evaluation is composed of two different videos, one acquired with a JAI's AD-080GE camera and the other with a TASE150 camera. Given the limitations of the OTB framework on evaluating videos with out-of-view targets, the dataset videos were split into smaller sequences that always have the target in the field-of-view. In total, the visible spectrum dataset has 8747 manually annotated frames. These smaller sequences include cases where a vessel goes through regions of sun reflections, cases of highly irregular wakes, and cases where the target deploys smaller vessels and life rafts. The LWIR dataset is also composed of two sequences obtained with a GOBI-384 camera. In total the infrared dataset is composed of 2422 manually annotated frames. These videos were obtained in three different days in the coastal area of Portimão. In Table 5.1 the description of each sequence is presented.

Table 5.1: Description of all sequences used to evaluate our method and other state-of-the-art tracking algorithms.

Video	Description	Frames	Date
TASE (Visible)	A medium sized vessel deploys a smaller vessel. Irregular wakes and waves. Fast motion.	1685	21-04-2015
JAI 1 (Visible)	A medium sized vessel deploys and retracts a smaller vessel. Irregular wakes and waves. Intense sun-reflections.	4384	22-04-2015
JAI 2 (Visible)	A medium sized vessel. Irregular wakes and waves. Intense sun-reflections.	297	22-04-2015
JAI 3 (Visible)	A medium sized vessel. Irregular wakes and waves.	101	22-04-2015
JAI 4 (Visible)	A medium sized vessel followed by a smaller vessel. Irregular wakes and waves.Intense sun-reflections.	2280	22-04-2015
GOBI 1 (LWIR)	Medium sized vessel. Fast motion.	1718	22-04-2015
GOBI 2 (LWIR)	Medium sized vessel. Fast motion.	704	02-06-2015

5.2.2 Data Labeling

To conduct the evaluations we require the real location of the target (ground truth) in each frame to compare with the detections obtained with the tracking methods. The OTB framework uses a rectangular bounding box of the target location to perform the evaluation. The labeling of the ground truth has to be performed manually because any error in these annotations will compromise the evaluation results. Due to the number of videos which have a high number of frames, this task is considered very time consuming and any efforts to facilitate it are welcome. This is why we use the labeling tool developed in the SEAGULL project [1]. This labeling tool has a simple interface and automates multiple actions to help in the annotation process.

The result of this labeling effort is made publicly available in order to incite the research in this field. The annotations can be accessed online[1] for further research and validation of the results presented below. See Fig. 5.1, Fig. 5.2 and Fig. 5.10 for some examples of frames from the dataset.

5.3 Tested Tracking Algorithms

We evaluate some of the state-of-the art tracking methods mentioned in Chapter 2, that were available online for testing purposes, using the OTB framework. The tested trackers were ASMS [22], DSST [11], KCF [7], CF2 [16], SRDCF [14], MUSTer [13], MEEM [21], MDNet [18], and the method

[1] http://vislab.isr.ist.utl.pt/seagull-dataset/

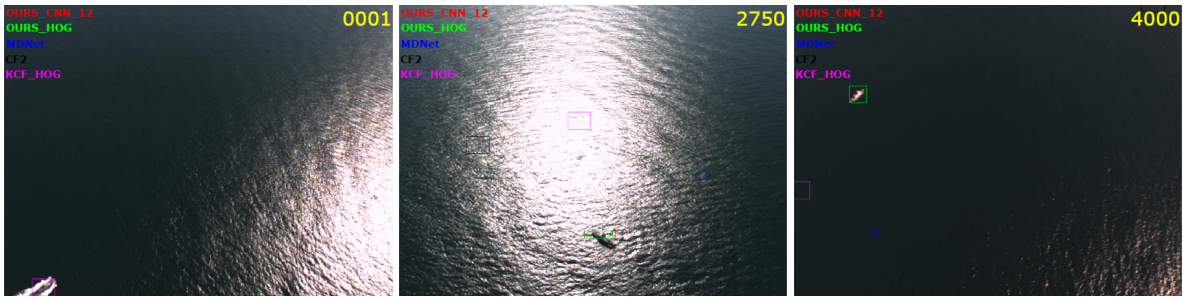


Figure 5.1: Frames from the JAI 1 sequence with the results, as bounding boxes, of some of the best performing methods: both our methods (CNN as red and HOG as green), MDNet (blue), CF2 (black) and KCF with HOG (pink). In the first frame (left) all trackers are initialized by the same BB which means only one is visible (KCF). This example is a case of success for both our methods given that the track is still on the target after 4000 frames, even after going through a region with intense sun-reflections multiple times. The other top performing algorithms lose the target much earlier.

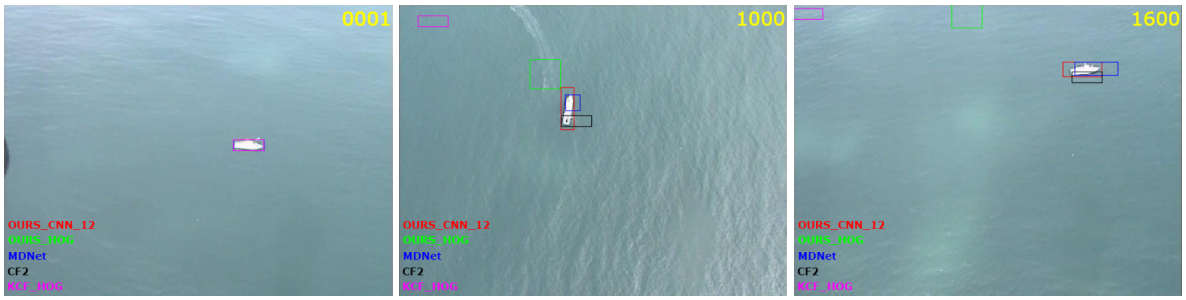


Figure 5.2: Frames from the TASE sequence with the results, as bounding boxes, of some of the best performing methods: both our methods (CNN as red and HOG as green), MDNet (blue), CF2 (black) and KCF with HOG (pink). In the first frame (left) all trackers are initialized by the same BB which means only one is visible (KCF). This example is a case of failure for our method using HOG features, where the track gets lost following the wake of another target. Nevertheless, our method using CNN features can overcome this problem given its capability to better discriminate the target from the wakes. Note that with this challenging sequence with multiple vessels, highly irregular wakes and fast motion only the trackers using CNN were able to keep tracking the target for the entire sequence.

proposed in this work, either using raw image pixels, HOG features or CNN features. We also evaluate extracting features from different layers of the VGG-Net in order to determine which better discriminate maritime vessels in the setting of this work.

5.4 Results

In this section we will show the results obtained using the OTB framework and the rate at which the different trackers process frames in Frames Per Second (FPS). We start by presenting the results of the image feature evaluation in the visible spectrum. Then we show the results of our methods compared to the original KCF either with HOG and the best CNN features. Next, the results of all the state-of-the-art and our proposed methods are presented. The same is done for the LWIR spectrum. Note that the colors in the following plots represent the ranking and not the different trackers. The same tracker can

have different colors in different evaluations depending on its ranking (i.e. red corresponds to the first place, green to the second place, blue to the third place and so on). This is a characteristic of the OTB framework. A video with some results of the top tracking methods on our dataset can be viewed online[2].

5.4.1 Visible Spectrum

The visible spectrum and the LWIR spectrum each have their own set of challenges. Due to this we perform the evaluations separately. First we present the results of the visible spectrum.

5.4.1.A Feature Selection for KCF on Visible Spectrum Images

In this section we present the results of the KCF tracker with different image features - Fig. 5.3 and Fig. 5.4. The evaluated features are raw image pixels, FHOG, and CNN from different layers of the VGG-Net. Right away we see that the CNN features outperform the others in the four evaluations, specially the layers 8 and 12, where the 12th slightly outperforms the 8th. The 16th layer has significantly less precision and success than the other two. This is due to how abstract the features from deeper layers are. In the setting of airborne maritime images the less abstract features show better results.

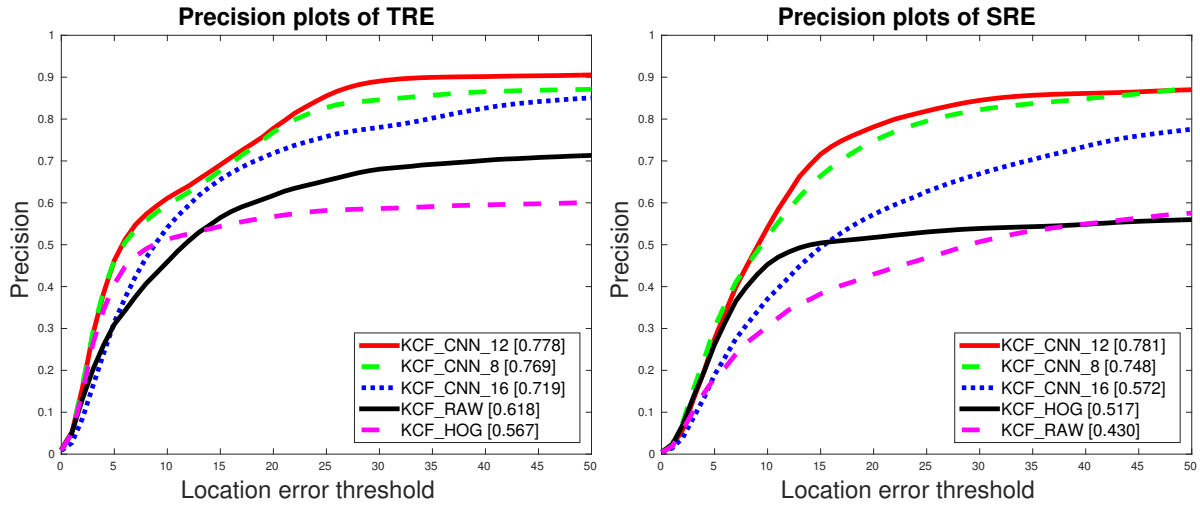


Figure 5.3: Visible spectrum. Precision plots of the SRE and the TRE of the KCF tracker using different features. The values on the legend correspond to the precision for a location error threshold of 20.

The raw image pixels overcome the FHOG features in both success and precision in the TRE, but the FHOG outperform in the SRE. From preliminary results we know that the raw image pixels are not robust to intense sun-reflections (track loses the target) but the FHOG allow the tracker to track in this situations. For these reasons we propose to use our method either with FHOG features or CNN features

[2]https://www.youtube.com/watch?v=oWQ5CuXC5DA

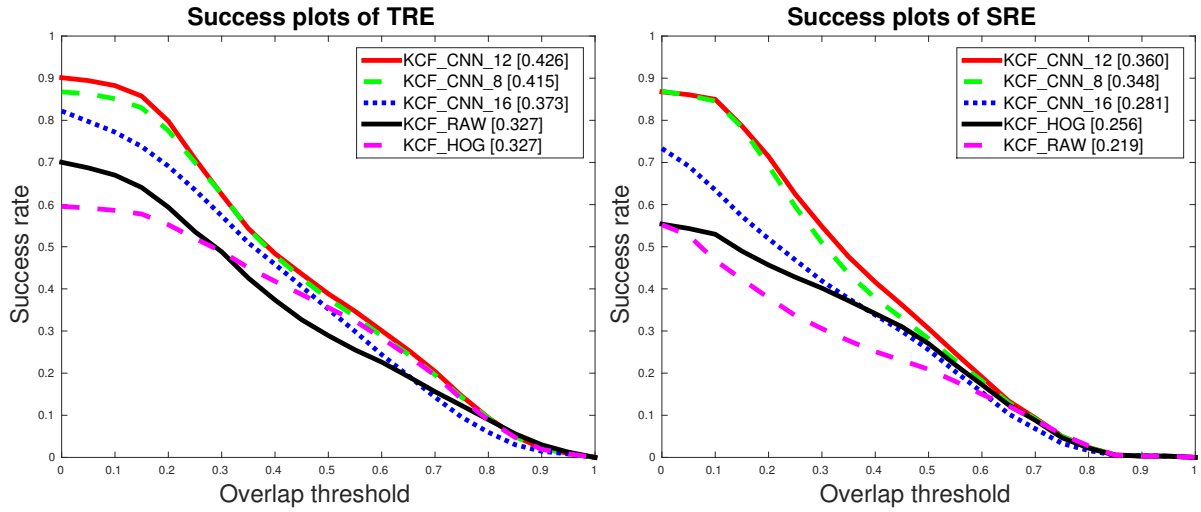


Figure 5.4: Visible spectrum. Success plots of the SRE and the TRE of the KCF tracker using different features. The values on the legend correspond to the area under the curve.

of the 12th layer due to their performance compared to the others. We conclude that the choice of image features has a significant effect in performance in the visible spectrum.

5.4.1.B Ours vs KCF

In this section we present the results of KCF and our method (*OURS*), which integrates the blob analysis step in the KCF tracker either using HOG features (*OURS_HOG*) or CNN features (*OURS_CNN*). We show these results in Fig. 5.5 and Fig. 5.6. These plots help visualize how much the Precision and Success increased when using our approach compared to using the basic KCF.

In the SRE our method (with HOG or CNN) outperforms the KCF, as expected. Note that even when using HOG our method outperforms KCF with CNN features. Comparing the methods using the same features we see the increase in Precision and Success that is consequence of the inclusion of the blob analysis step. In the Precision plots of SRE only 51.7% of the frames have correct estimations (considering a location error threshold of 20 pixels) when using the basic KCF while incorporating our blob analysis step increases this percentage to 85.2%. In the Success plot of SRE the AUC increases 56.64% for the same circumstances. In the case of CNN the increases in Precision and Success are not as pronounced as the HOG case but they are still significant. In the Precision plots of SRE the KCF with CNN features correctly detects 78.1% of the frames while ours with CNN correctly detects 98.2% for the same error threshold (20 pixels). In the Success plot of SRE the AUC increases 47.66% from the basic KCF with CNN features to our method using CNN features.

In the TRE we obtain similar results except that for some specific thresholds the basic KCF with CNN outperforms our method with HOG. Nevertheless our method with HOG outperforms in the most meaningful thresholds of 20 pixels for the location error and 0.5 for the bounding box overlap ratio [6]. In

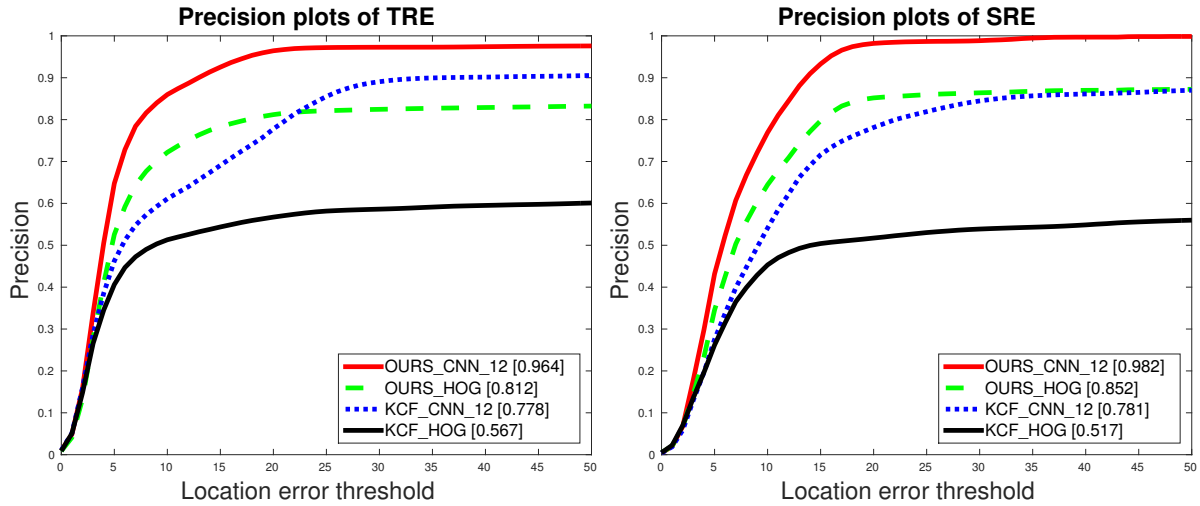


Figure 5.5: Visible spectrum. Precision plots of the SRE and the TRE of our method using two different types of images features and the basic KCF using the same features: HOG and CNN. The values on the legend correspond to the precision for a location error threshold of 20.

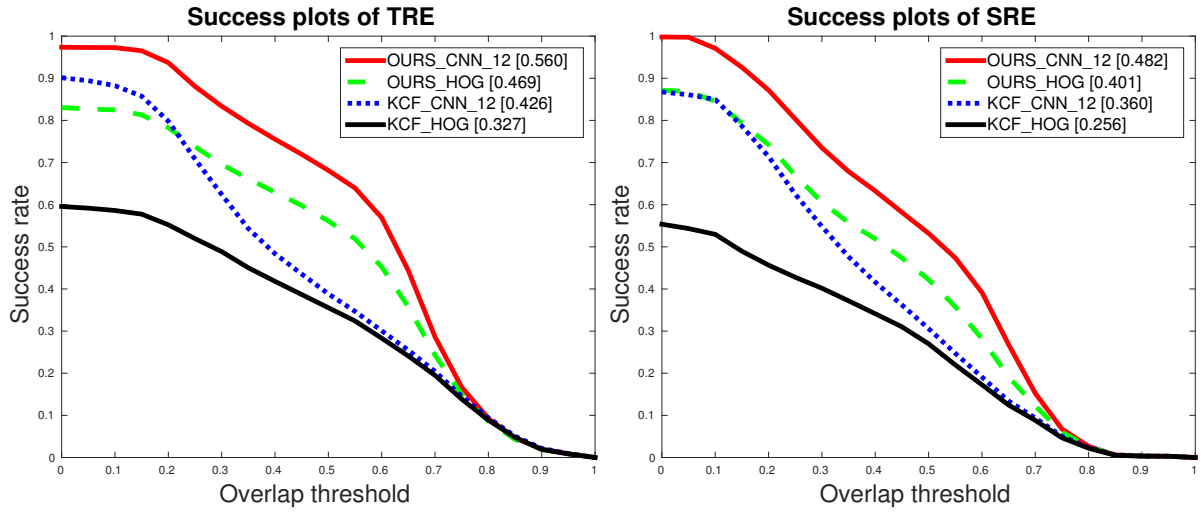


Figure 5.6: Visible spectrum. Success plots of the SRE and the TRE of our method using two different types of images features and the basic KCF using the same features: HOG and CNN. The values on the legend correspond to the area under the curve.

the Precision plots of TRE, KCF with HOG only estimates correctly 56.7% of the frames while ours with HOG estimates correctly 81.2%. It is interesting to note that our method has better Precision in the SRE than in the TRE. Remember that these evaluations are independent and it is not supposed to expect better temporal robustness just because in the TRE the bounding box has no spacial error. Even tough we can't conclude much from comparing the TRE and SRE we can conclude that our method is robust to both of these evaluations independently. It is specially robust to the SRE because of the blob analysis step which corrects the error introduced in the initialization. In the Precision plots the KCF with CNN jumps from 77.8% of correct estimations to 96.4% when using our approach with CNN. In the Success

plot of TRE there is an increase of 55.47% from the KCF with HOG to ours with HOG and an increase of 52.34% from the KCF with CNN to ours with CNN.

We conclude, from Fig. 5.5 and Fig. 5.6, that our method, either using CNN or HOG features, outperforms the basic KCF with either features, as expected.

5.4.1.C Ours vs All

Now that we have shown the increase in robustness by incorporating the blob analysis step and evaluated different image features in the maritime scenario from an airborne perspective, we will show the results from the multiple state-of-the-art general purpose tracking methods mentioned in Sec. 5.3.

In Fig. 5.7 and Fig. 5.8 we show the results for all evaluated trackers. The results show that our method using CNN features outperforms all the others in the four different evaluations. Our method using HOG features is the second in three of them, only being third in the Success plots of SRE where the MDNet tracker takes the second place. As stated in Chapter 2 the MDNet is the most recent winner (2015) of the acclaimed VOT challenge [9]. In the Precision plots of SRE our method with CNN correctly estimates the location of the target (i.e. with an error below 20 pixels) in 96.4% of the frames while ours with HOG estimates correctly 81.2%. In third is the MDNet tracker with 79.2%. Similar results were obtained in the SRE. Our proposed methods show incredible precision in the visible spectrum of the airborne maritime scenario when compared with other tracking methods thanks to the blob analysis step. This is even more significant when considering the number of FPS that our method with HOG features can compute (see Table 5.2). Our method with CNN features averages 2.39 FPS, our method with HOG averages 32.77 FPS and the MDNet averages 0.37 FPS.

In the Success plots of TRE (Fig. 5.8) our method with HOG features has an increase of 1.01%

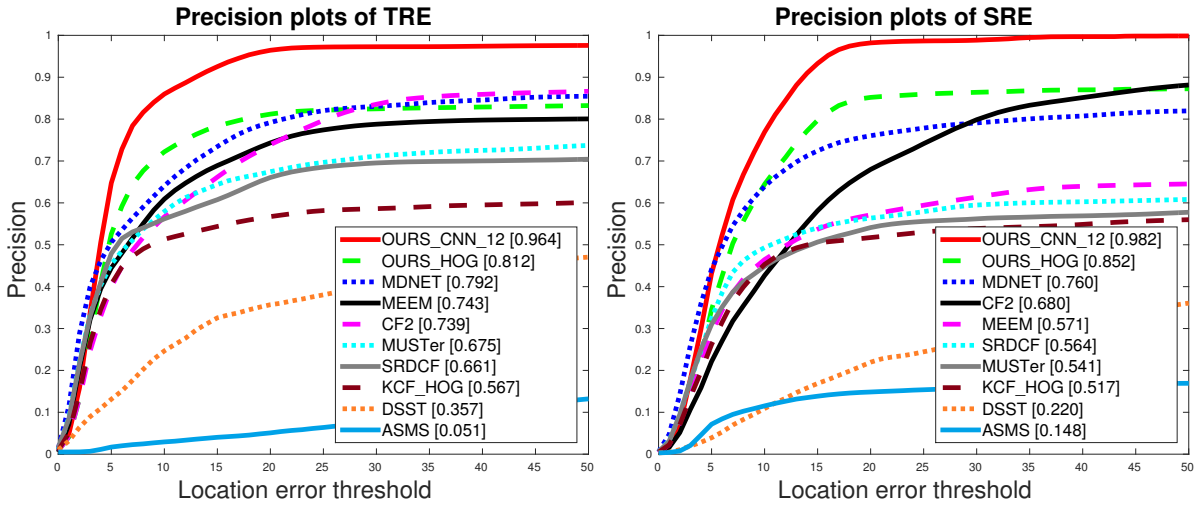


Figure 5.7: Visible spectrum. Precision plots of the SRE and the TRE with our methods and state-of-the-art tracking methods. The values on the legend correspond to the precision for a location error threshold of 20.

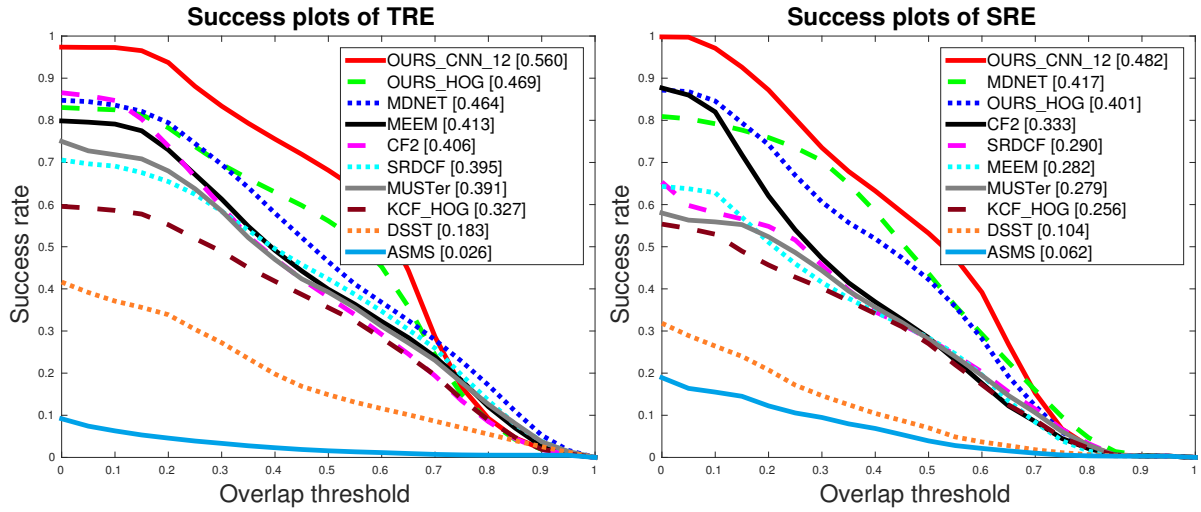


Figure 5.8: Visible spectrum. Success plots of the SRE and the TRE with our methods and state-of-the-art tracking methods. The values on the legend correspond to the area under the curve.

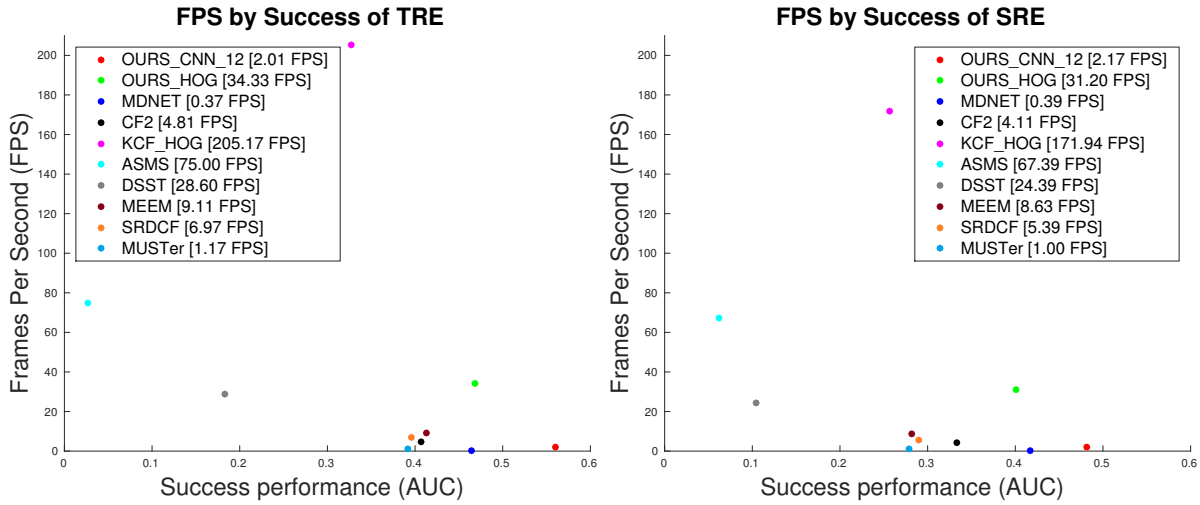


Figure 5.9: FPS by Success of TRE and SRE of all the evaluated tracking methods. Visible spectrum.

compared with the MDNet. Ours with CNN has an increase of 20.69%. In the Success plots of SRE our method with HOG has a decrease of 3.84% with respect to MDNet but ours with CNN has an increase of 15.59%. Most trackers do not adapt the bounding box to the tracked target. Usually the bounding box width and height stay constant. In the DSST and ASMS the scale change is estimated but the aspect ratio (width/height) stays the same. In our setting, when the target changes direction, its aspect ratio changes due to the top down perspective. The only trackers that are able to adjust the bounding box to the target are both our methods and the MDNet. This explains why they show better results than the others in the Success evaluation.

Note that some trackers that have great results in general purpose tracking challenges and benchmarks seem to not generalize well enough for the particular case of airborne maritime imagery, such as

the ASMS and the DSST. The DSST, even tough being a part of the correlation filter tracking family, does not use the kernel trick and the scale space search does not work well for the rotations present in this scenario, as explained in the previous paragraph.

To conclude, we have shown that our method has better performance than the state-of-the-art (MDNet) in the airborne maritime setting in both Precision and Success of the SRE and the TRE. Our method using CNN features is the top performing in all evaluations and runs at 2.39 FPS in average. Our method using HOG outperforms the MDNet in almost all evaluations, but processes frames at a rate almost three orders of magnitude faster than MDNet: 32.77 FPS and 0.37 FPS respectively. See Fig. 5.9 where we show the relation between the frame rate (FPS) of each tracker and its Success ranking.

5.4.2 Long-Wave Infrared Spectrum

In the following sections the results for the LWIR spectrum (Fig 5.10) will be shown. Similar to how we did for the visible spectrum, first we evaluate the different image features in the airborne maritime setting, now in the LWIR spectrum. Then we show the increase in performance obtained by integrating the blob analysis step in the KCF tracker. To conclude we show the results of the tested state-of-the-art tracking methods and their processing rate in FPS.

5.4.2.A Feature Selection for KCF on LWIR Images

In the LWIR the CNN features are still the top performing, specially the 12th layer of the VGG-Net (Fig. 5.11 and Fig. 5.12). What changed is that the raw image pixels have better performance overall than the HOG features in this spectrum. This seems to be the case because in this spectrum the challenges of sun reflections and irregular waves and wakes are not present. The advantage of the HOG features in

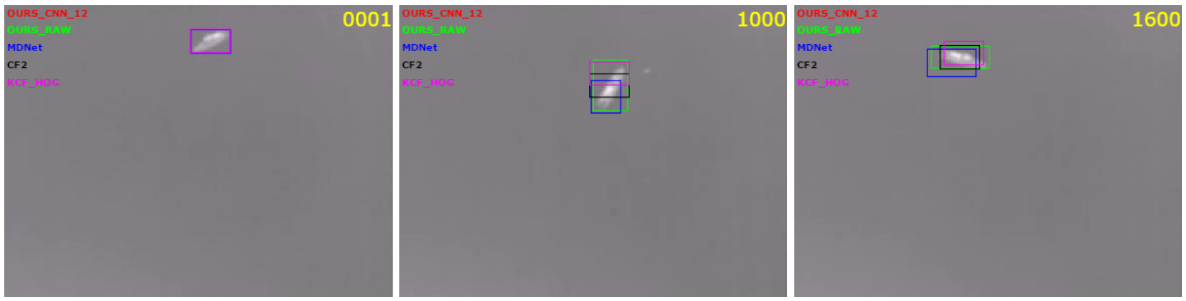


Figure 5.10: Frames from the GOBI 1 sequence with the results, as bounding boxes, of some of the best performing methods: both our methods (CNN as red and RAW as green), MDNet (blue), CF2 (black) and KCF with HOG (pink). In the first frame (left) all trackers are initialized by the same BB which means only one is visible (KCF). In this example the main challenges such as sun reflections and wakes are not present and this is why all the top trackers were able to track the target. Nevertheless, our method still has the advantage of adjusting the bounding box to the target with the blob analysis step. The MDNet bounding box regression is not as robust as our method in this scenario. The other trackers maintain a bounding box with constant size.

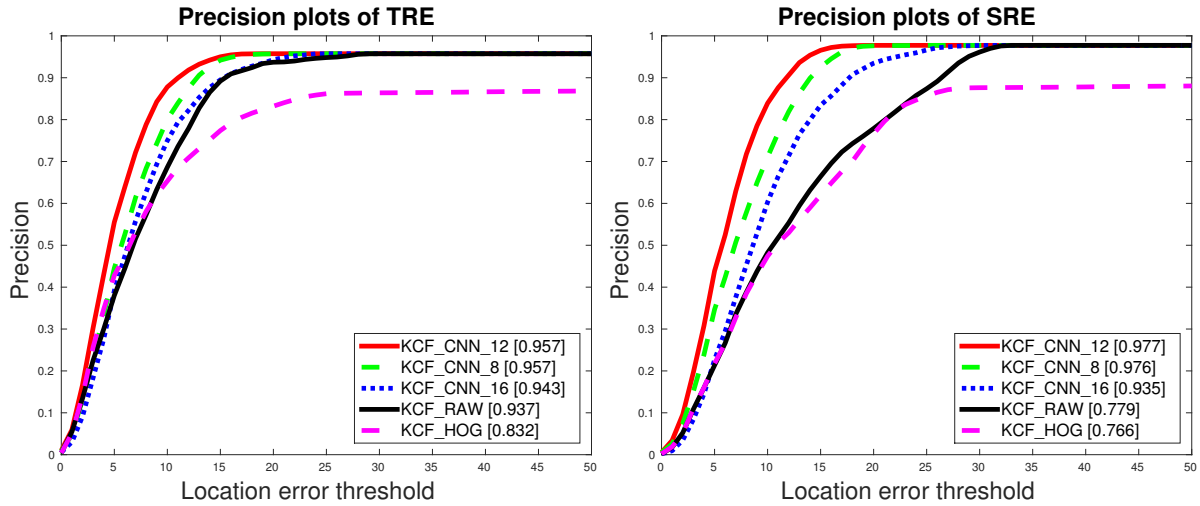


Figure 5.11: LWIR spectrum. Precision plots of the SRE and the TRE of the KCF tracker using different features. The values on the legend correspond to the precision for a location error threshold of 20.

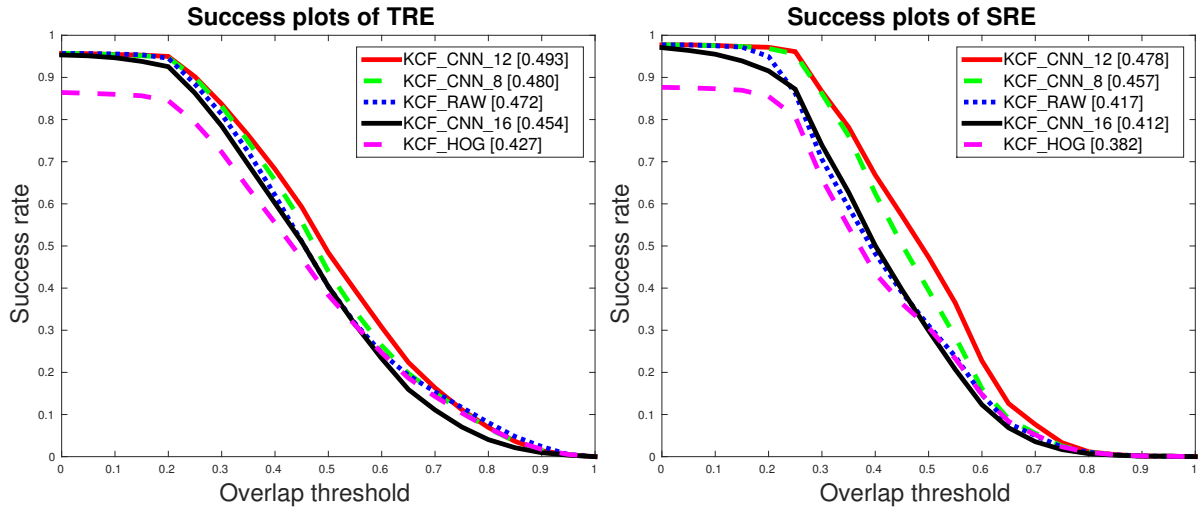


Figure 5.12: LWIR spectrum. Success plots of the SRE and the TRE of the KCF tracker using different features. The values on the legend correspond to the area under the curve.

the visible spectrum was the preprocessing that decreased the negative influence of these challenging situations. Due to the better performance of the raw image pixels, the following evaluations of our method are done with raw image pixels and CNN features from the 12th layer. Also, we can see that the choice of image features is not as significant in the LWIR spectrum compared with the visible spectrum.

5.4.2.B Ours vs KCF

The comparison between the proposed method and the KCF algorithm is shown in Fig. 5.13 and Fig. 5.14. In the Precision evaluation all methods show similar results, except the basic KCF with raw image pixels as input, which shows significant worse performance. Both our methods outperform the

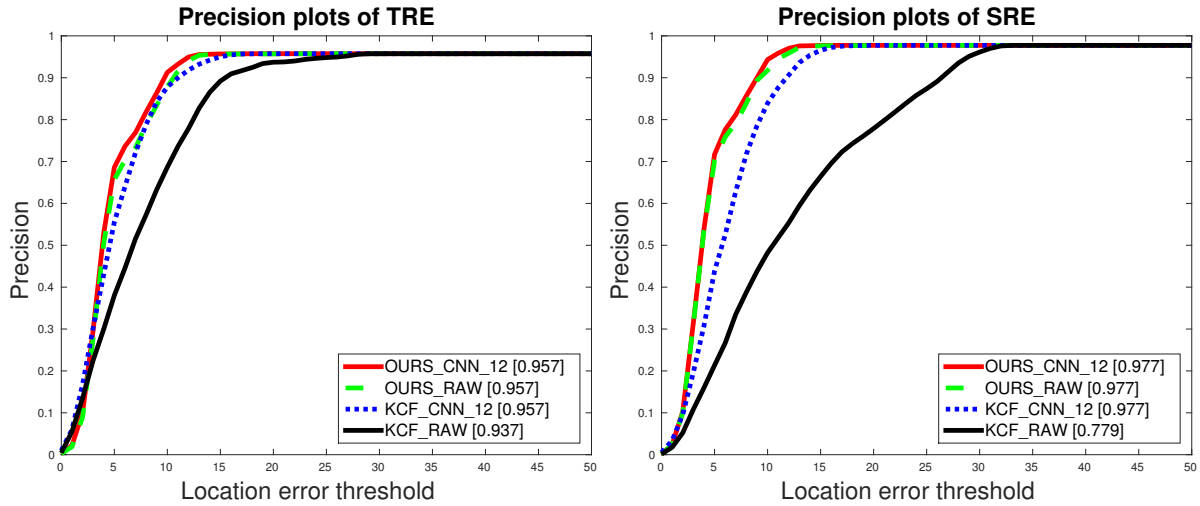


Figure 5.13: LWIR spectrum. Precision plots of the SRE and the TRE of our method using two different types of images features and the basic KCF using the same features: raw and CNN. The values on the legend correspond to the precision for a location error threshold of 20.

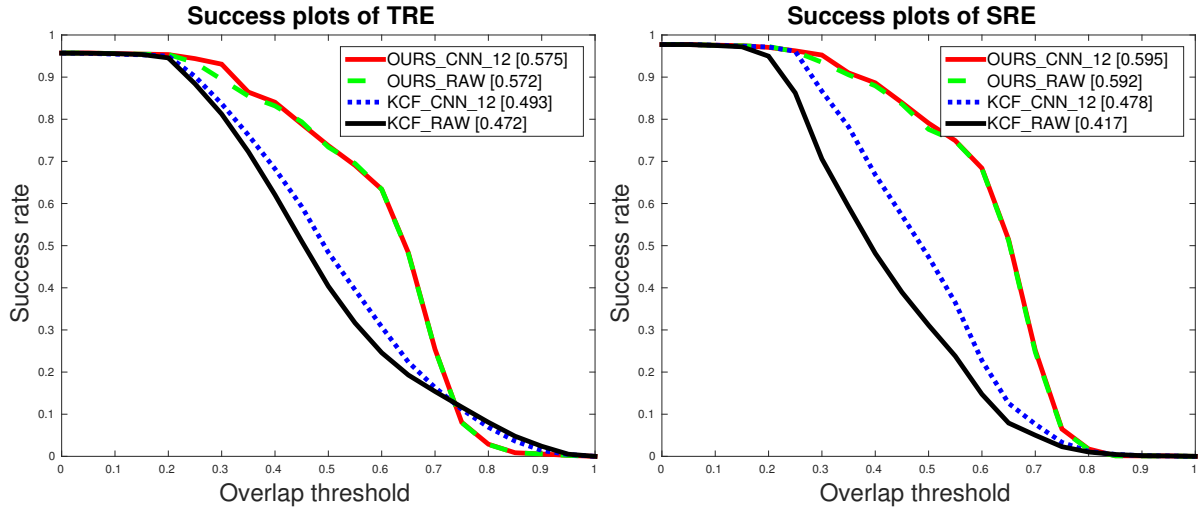


Figure 5.14: LWIR spectrum. Success plots of the SRE and the TRE of our method using two different types of images features and the basic KCF using the same features: raw and CNN. The values on the legend correspond to the area under the curve.

basic KCF by analyzing the different thresholds. In the Success evaluation both our methods significantly outperform both KCF (raw and CNN). This is due to the way our method adjust the bounding box to the target using the detected blob while the width and height in the basic KCF are constant. Furthermore, both our methods have very similar results in all thresholds of all evaluations. This happens because the LWIR spectrum is a less challenging scenario compared with the visible spectrum and the more complex preprocessing of the CNN is not necessary in this spectrum.

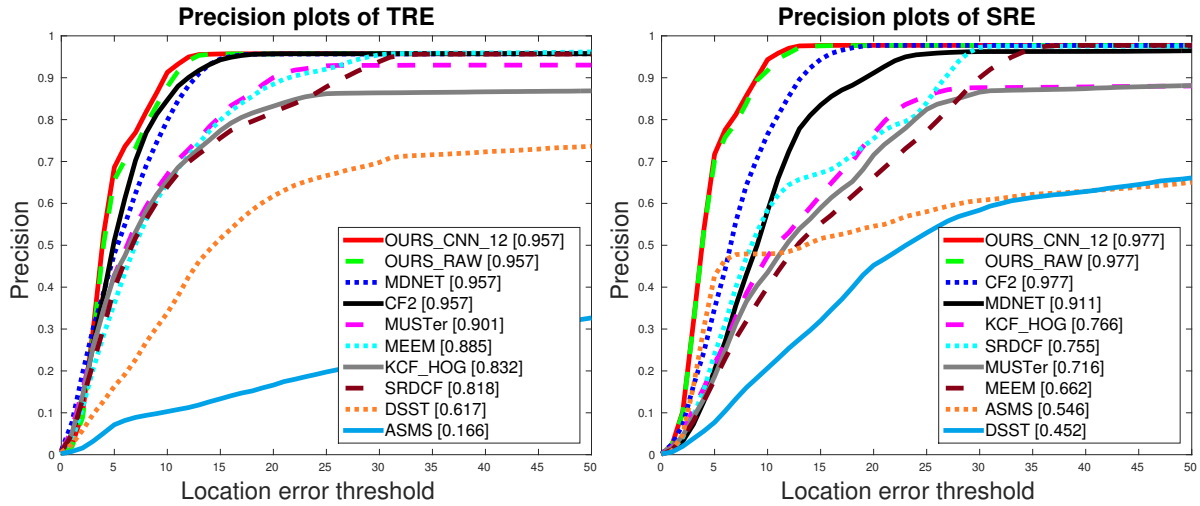


Figure 5.15: LWIR spectrum. Precision plots of the SRE and the TRE with our methods and state-of-the-art tracking methods. The values on the legend correspond to the precision for a location error threshold of 20.

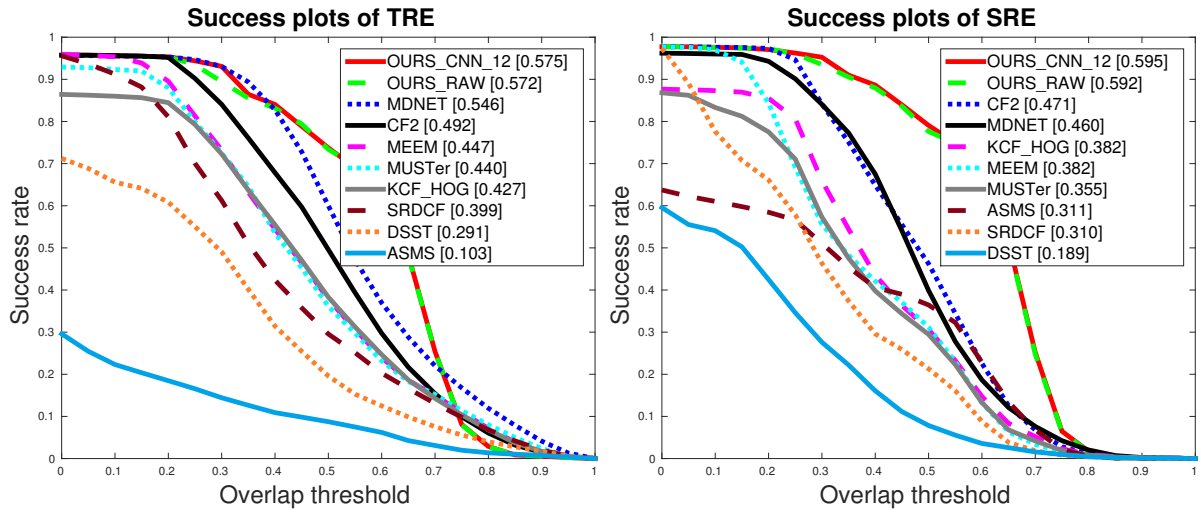


Figure 5.16: LWIR spectrum. Success plots of the SRE and the TRE with our methods and state-of-the-art tracking methods. The values on the legend correspond to the area under the curve.

5.4.2.C Ours vs All

In Fig. 5.15 and Fig. 5.16 we show the results of our methods and other state-of-the-art tracking methods in the LWIR spectrum. Once more our methods, either using raw image pixels or CNN features, outperform the other methods in every evaluation. This is the case since the blob analysis step is specially robust because of the Otsu's method, which works very well in this spectrum (the frames are always bimodal if the target is present). The SRE results are once more evidence of the robustness of our method to spatial error in the initialization because of how much the Precision and Success increase compared to the other methods, which is not as significant in the TRE.

The other methods rank similarly with the visible spectrum relative to one another, but in absolute

terms they have better performance in the LWIR as expected due to the lack of the most challenging situations present in the visible spectrum such as sun reflections and wakes.

See Fig. 5.9 where we show the relation between the frame rate (FPS) of each tracker and its Success ranking. In Table 5.2 we also show the average frame rate of each tracker in this spectrum. The frame rate is similar to the visible spectrum for the majority of the trackers. The exception are those that depend on the image resolutions and are faster due to the lower resolution of the GOBI camera.

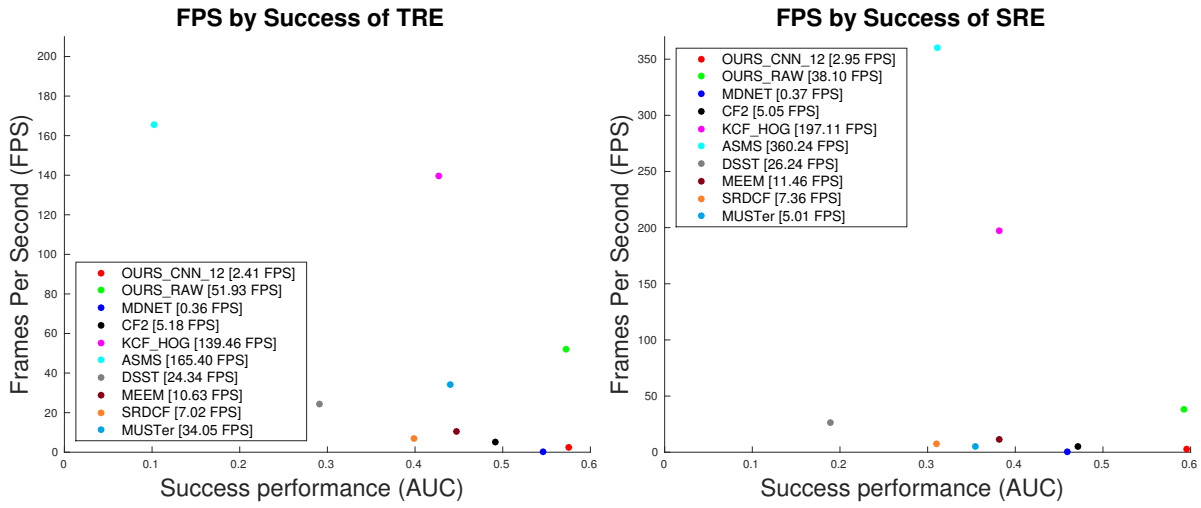


Figure 5.17: FPS by Success of TRE and SRE of all the evaluated tracking methods. Visible spectrum.

Table 5.2: The used programming language and the frame rate of each evaluated tracking method in all evaluations for both the visible spectrum and LWIR spectrum and the average over all the evaluations.

		FPS (Visible)		FPS (LWIR)		
Method	**Implementation**	**TRE**	**SRE**	**TRE**	**SRE**	**Average**
Ours_CNN_12	Python	2.01	2.17	2.41	2.95	2.39
Ours_HOG	Python	34.33	31.20	NA	NA	32.77
Ours_HOG	Python	NA	NA	51.93	38.10	45.02
MDNet	Matlab	0.37	0.39	0.36	0.37	0.37
CF2	Matlab	4.81	4.11	5.18	5.05	4.79
KCF_HOG	Matlab	205.17	171.94	139.46	197.11	178.42
ASMS	C++	75.00	67.39	165.40	360.24	167.01
DSST	Matlab	28.60	24.39	24.34	25.24	25.64
MEEM	Matlab	9.11	8.63	10.63	11.46	9.97
SRDCF	Matlab	6.97	5.39	7.02	7.36	6.69
MUSTer	Matlab	1.17	1.00	4.54	5.01	2.93

5.4.3 Limitations

Our proposed method still has some limitations in the visible spectrum and the LWIR spectrum. The Precision results are usually better than the Success results overall. This points to a problem with all trackers, which is the bounding box estimation. As previously stated, the bounding box size stays constant in most of the trackers. The proposed method is able to tackle this problem to some extend with the blob analysis step but cannot adjust the bounding box when the blob analysis is not being used (such as with sun reflections).

Other limitations that our method still has is tracking of fast motions (usually due to the camera being controlled by the base station operator) and targets with very low resolution (small targets such as life rafts).

Also, the correlation filter is updated for a long time (thousands of frames) and a lot of noise is accumulated over time. The robustness could be improved by including a method to reinitialize the KCF after some time and when there is a high confidence on the target's estimated location.

In the LWIR some of these limitations are not applicable due to the nature of this spectrum which filters out sun reflections, and wakes. The bounding box estimation has better results due to this. Fast motion and small targets are still limitations to our method in this spectrum.

The other main limitation, is that there is no mechanism to determine if the tracker is lost and there is no re-detection process, which is specially pertinent when the target leaves the field of view. Also multi-target tracking was not specifically tackled. Running multiple instances of the proposed tracker should work in the general case due to the nearest neighbor search in the blob analysis.

6

Conclusion & Future Work

In this work we presented a new algorithm for tracking that overcomes some of the main concerns in visual tracking in airborne oceanographic imagery, namely sun-reflections, scale changes and long term-tracking. These challenging problems are overcome thanks to a combination of correlation filters and blob analysis. The presence of background clutter such as waves and wakes was also overcome to some extend, except in the presence of fast motions where the tracker can wrongly estimate the location and size of the target.

We present two versions of the algorithm. The best performing is based on CNN features and significantly outperforms current state-of-the-art general purpose trackers. The second version uses HOG features, in the visible spectrum, and raw image pixels in the LWIR spectrum, and attains performances competitive with the current state-of-the-art, but at a much faster frame-rate, suitable for real-time operation on airborne systems.

The main limitations are the following. a) the bounding box adjustment only works during the blob analysis step since it is dependent on the quality of the image segmentation; b) Failure when tracking small targets, which have very low resolution; c) the KCF tracker is never reinitialized, which means it will keep accumulating error for a long time; d) there is no mechanism to determine tracking failure, and no re-detection module and e) there is no dedicated mechanism for multi-target tracking.

For future work, it is proposed the fusion of a detection module with the tracking step proposed here to allow for a completely autonomous system. This would also include a failure detection module as well as the use of data association techniques for multiple target tracking.

As we have seen, in the visible spectrum, the choice of image features has a significant effect in the Precision and Success results. Due to this, we propose the evaluation of other features or even using different features simultaneously in the airborne maritime scenario. Furthermore, it is worth evaluating the performance of the features extracted from a CNN fine-tuned to this setting.

Improvements on the bounding box and target segmentation could increase the precision and success rate of the algorithm. A possible path of research could focus on temporal filtering during blob analysis. This consists on only accepting a detection if it persist through time (couple of frames). Also, instead of using heuristic rules to determine if the image segmentation has quality enough to recenter the target we propose to train a binary classifier (such as a SVM) with that end. Given enough training data, this method should be able to better discriminate both situations compared with the hand-crafted conditions proposed.

In conclusion we had a challenging problem and used a challenging dataset from a real world scenario. Our method using CNN outperforms all tracking methods in the four evaluations. Our method using HOG or raw image pixels is competitive with state-of-the-art but processes frames three orders of magnitude faster. The code and the dataset used are publicly available online for further research.

Bibliography

[1] M. M. Marques, P. Dias, N. P. Santos, V. Lobo, R. Batista, D. Salgueiro, A. Aguiar, M. Costa, J. E. da Silva, A. S. Ferreira, J. Sousa, M. de Fátima Nunes, E. Pereira, J. Morgado, R. Ribeiro, J. S. Marques, A. Bernardino, M. Griné, and M. Taiana, "Unmanned aircraft systems in maritime operations: Challenges addressed in the scope of the seagull project," in *OCEANS 2015 - Genova*, May 2015, pp. 1–6.

[2] V. Anastasia and C. Popescu, "Piracy in the gulf of aden - a problem of our days," *Constanta Maritime University Annals*, vol. 13, no. 1, pp. 45–50, 2010.

[3] H. K. Kang, "Gulf of aden vs malacca strait: Piracy and counter-piracy efforts," *Institute of Peace and Conflict Studies*, 2009.

[4] N. Mathiason, V. Parsons, and T. Jeory, "Europe's refugee crisis: Is frontex bordering on chaos?" http://labs.thebureauinvestigates.com/is-frontex-bordering-on-chaos/, accessed: 21-12-2015.

[5] J. S. Marques, A. Bernardino, G. Cruz, and M. Bento, "An algorithm for the detection of vessels in aerial images," in *Advanced Video and Signal Based Surveillance (AVSS), 2014 11th IEEE International Conference on*. IEEE, 2014, pp. 295–300.

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

[7] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista, "High-speed tracking with kernelized correlation filters," *Pattern Analysis and Machine Intelligence, IEEE Trans.*, vol. 37, no. 3, pp. 583–596, 2015.

[8] J. Matos, A. Bernardino, and R. Ribeiro, "Robust tracking of vessels in oceanographic airborne images," in *OCEANS'16 MTS/IEEE Monterey*. MTS/IEEE.

[9] "The visual object tracking vot2015 challenge results," Dec 2015. [Online]. Available: http://www.votchallenge.net/vot2015/

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

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

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

[13] Z. Hong, Z. Chen, C. Wang, X. Mei, D. Prokhorov, and D. Tao, “Multi-store tracker (muster): A cognitive psychology inspired approach to object tracking,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 749–758.

[14] M. Danelljan, G. Hager, F. Shahbaz Khan, and M. Felsberg, “Learning spatially regularized correlation filters for visual tracking,” in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 4310–4318.

[15] M. Danelljan, G. Hager, F. Khan, and M. Felsberg, “Convolutional features for correlation filter based visual tracking,” in *Proceedings of the IEEE Int. Conf. on Computer Vision Workshops*, 2015, pp. 58–66.

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

[17] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman, “Return of the devil in the details: Delving deep into convolutional nets,” *arXiv preprint arXiv:1405.3531*, 2014.

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

[19] N. Wang and D.-Y. Yeung, “Learning a deep compact image representation for visual tracking,” in *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2013, pp. 809–817.

[20] S. Hare, S. Golodetz, A. Saffari, V. Vineet, M.-M. Cheng, S. Hicks, and P. Torr, “Struck: Structured output tracking with kernels,” 2014.

[21] J. Zhang, S. Ma, and S. Sclaroff, “MEEM: robust tracking via multiple experts using entropy minimization,” in *Proc. of the European Conference on Computer Vision (ECCV)*, 2014.

[22] T. Vojir, J. Noskova, and J. Matas, “Robust scale-adaptive mean-shift for tracking,” *Pattern Recognition Letters*, vol. 49, pp. 250–258, 2014.

[23] M. Teutsch and W. Krüger, “Classification of small boats in infrared images for maritime surveillance,” in *2010 International WaterSide Security Conference*. IEEE, 2010, pp. 1–7.

[24] G. Cruz and A. Bernardino, “Image saliency applied to infrared images for unmanned maritime monitoring,” in *International Conference on Computer Vision Systems*. Springer, 2015, pp. 511–522.

[25] G. Mattyus, “Near real-time automatic vessel detection on optical satellite images,” in *ISPRS Hannover Workshop*. ISPRS Archives, 2013, pp. 233–237.

[26] D. Bloisi, L. Iocchi, M. Fiorini, and G. Graziano, “Automatic maritime surveillance with visual target detection,” in *Proc. of the International Defense and Homeland Security Simulation Workshop (DHSS)*, 2011, pp. 141–145.

[27] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *CoRR*, vol. abs/1409.1556, 2014.

[28] D. S. Bolme, , B. A. Draper, and J. R. Beveridge, “Average of synthetic exact filters,” in *Computer Vision and Pattern Recognition (CVPR), 2009 IEEE Conference on*. IEEE, 2009, pp. 2105–2112.

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

[30] B. Schölkopf and A. J. Smola, *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2002.

[31] R. M. Gray, *Toeplitz and circulant matrices: A review*. now publishers inc, 2006.

[32] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista, “Exploiting the circulant structure of tracking-by-detection with kernels,” in *European conference on computer vision*. Springer, 2012, pp. 702–715.

[33] S. Chaudhuri, R. Velmurugan, and R. Rameshan, “Mathematical background,” in *Blind Image Deconvolution: Methods and Convergence*. Springer, 2014, pp. 11–35.

[34] P. C. Hansen, J. G. Nagy, and D. P. O’leary, *Deblurring images: matrices, spectra, and filtering*. Siam, 2006, vol. 3.

[35] R. B. Blackman and J. W. Tukey, “The measurement of power spectra,” 1958.

[36] E. R. Kanasewich, *Time sequence analysis in geophysics*. University of Alberta, 1981.

[37] M. Danelljan, F. Shahbaz Khan, M. Felsberg, and J. Van de Weijer, “Adaptive color attributes for real-time visual tracking,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 1090–1097.

[38] N. Dalal and B. 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.

[39] X. Glorot, A. Bordes, and Y. Bengio, “Deep sparse rectifier neural networks.” in *Aistats*, vol. 15, no. 106, 2011, p. 275.

[40] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “ImageNet: A Large-Scale Hierarchical Image Database,” in *CVPR09*, 2009.

[41] N. Otsu, “A threshold selection method from gray-level histograms,” *Automatica*, vol. 11, no. 285-296, pp. 23–27, 1975.

[42] S. Suzuki *et al.*, “Topological structural analysis of digitized binary images by border following,” *Computer Vision, Graphics, and Image Processing*, vol. 30, no. 1, pp. 32–46, 1985.

[43] K. F. Riley, M. P. Hobson, and S. J. Bence, *Mathematical methods for physics and engineering: a comprehensive guide*. Cambridge University Press, 2006.

[44] L. Yang and F. Albregtsen, “Fast and exact computation of cartesian geometric moments using discrete green’s theorem,” *Pattern Recognition*, vol. 29, no. 7, pp. 1061–1073, 1996.

[45] R. C. Gonzalez and R. E. Woods, “Digital image processing,” 2008.

[46] D. E. King, “Dlib-ml: A machine learning toolkit,” *Journal of Machine Learning Research*, vol. 10, pp. 1755–1758, 2009.

[47] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, “Caffe: Convolutional architecture for fast feature embedding,” *arXiv preprint arXiv:1408.5093*, 2014.

[48] G. Bradski, “The OpenCV Library,” *Dr. Dobb’s Journal of Software Tools*, 2000.

[49] B. Babenko, M.-H. Yang, and S. Belongie, “Robust object tracking with online multiple instance learning,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 8, pp. 1619–1632, 2011.