Carnegie Mellon University

CARNEGIE INSTITUTE OF TECHNOLOGY

THESIS

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF Doctor of Philosophy

TITLE **Accounting for Aliasing in Correlation Filters:**

Zero-Aliasing and Partial-Aliasing Correlation Filters

PRESENTED BY **Joseph A. Fernandez**

ACCEPTED BY THE DEPARTMENT OF

Electrical and Computer Engineering

___Vijayakumar Bhagavatula_____ _____4/30/14_____
ADVISOR, MAJOR PROFESSOR DATE

___Jelena Kovacevic_____ ____4/30/14_____
DEPARTMENT HEAD DATE

APPROVED BY THE COLLEGE COUNCIL

___Vijayakumar Bhagavatula_____ _____4/30/14_____
DEAN DATE

Accounting for Aliasing in Correlation Filters: Zero-Aliasing and Partial-Aliasing Correlation Filters

Submitted in partial fulfillment of the requirements for

the degree of

Doctor of Philosophy

in

Electrical and Computer Engineering

Joseph A. Fernandez

B.S., Electrical Engineering, New Mexico Institute of Mining and Technology

M.S., Electrical and Computer Engineering, Carnegie Mellon University

Carnegie Mellon University

Pittsburgh, PA

May 2014

ABSTRACT

Correlation filters (CFs) are well established and useful tools for a variety of tasks in signal processing and pattern recognition, including automatic target recognition and tracking, biometrics, landmark detection, and human action recognition. Traditionally, CFs have been designed and implemented efficiently in the frequency domain using the discrete Fourier transform (DFT). However, the element-wise multiplication of two DFTs in the frequency domain corresponds to a circular correlation, which results in aliasing (i.e., distortion) in the correlation output. Prior CF research has largely ignored these aliasing effects by making the assumption that linear correlation is approximated by circular correlation. In this work, we investigate in detail the topic of aliasing in CFs. First, we illustrate that the current formulation of CFs in the frequency domain is inherently flawed, as it unintentionally assumes circular correlation during the design phase. This means that existing CFs are not truly optimal. We introduce *zero-aliasing correlation filters* (ZACFs) which fix this formulation issue by ensuring that each CF formulation problem corresponds to a linear correlation rather than a circular correlation. By adopting the ZACF design modifications, we show that the recognition and localization performance of conventional CF designs can be significantly improved. We demonstrate these benefits using a variety of data sets and present solutions to the computational challenges associated with computing ZACFs.

After a CF is designed, it is used for object recognition by correlating it with a test signal. We investigate the use of the well-known overlap-add (OLA) and overlap-save (OLS) algorithms to improve the computation and memory requirements of this correlation operation for high dimensional applications (e.g., video). Through this process, we highlight important tradeoffs between

these two algorithms that have previously been undocumented. To improve the computation and memory requirements of OLA and OLS, we introduce a new block filtering scheme, denoted partial-aliasing OLA (PAOLA) that intentionally introduces aliasing into the output correlation. This aliasing causes conventional CFs to perform poorly. To remedy this, we introduce *partial-aliasing correlation filters* (PACFs), which are specifically designed to minimize this aliasing. We demonstrate through numerical results that PACFs outperform conventional CFs in the presence of aliasing.

ACKNOWLEDGMENTS

First, I would like to acknowledge the source for my research funding. The first two and a half years of my graduate studies was funded by the Electrical and Control Integration Laboratory, General Motors (GM) Corporation. The funding for the bulk of this work was provided by the United States Department of Defense (DoD) through the National Defense Science & Engineering Graduate Fellowship (NDSEG) Program. Finally, I have also received financial support from King Abdulaziz City for Science and Technology (KACST), Saudi Arabia.

I would like to thank my advisor, Professor Vijayakumar Bhagavatula, for his incredible advice, guidance, and insight into my research. Kumar's ability to explain any topic and turn confusion into immediate clarity is unparalleled. I would also like to thank my former advisor, Professor Daniel D. Stancil (now Department Head at North Carolina State University) for introducing me to research and advising me for the first two years of my graduate student tenure. I have been truly blessed with two of the best advisors a graduate student can ask for. I would also like to thank the members of my committee, Marios Savvides, Richard Stern, and Abhijit Mahalanobis for their interest and excitement about my research. I would like to thank Andres Rodriguez and Vishnu Naresh Boddeti for contributing to this work and for their useful insights, suggestions, and feedback. I would also like to thank Kathy Brigham, Jon Smereka, and Stephen Siena for having useful conversations with me that have helped me overcome various hurdles over the years.

I would like to thank my officemates Jim Downey and Reginald Cooper for providing me with tremendous friendship, useful distractions, and many enlightening conversations about re-

search, cars, gadgets, or whatever else was on our minds. I am quite convinced that I have been blessed with the best officemates I could ask for. I would also like to thank all of my friends at the Newman Center and in the ECE department for your companionship over the years; the Gigahurtz softball team; the Fast Floorier Transforms floor hockey team; the Incapacitors flag football team; and the High-Z D basketball team. It has been a blessing to know each and every one of you and I will miss spending my time with you all.

I would like to thank my parents for everything they have done for me over the years. Your constant support, encouragement, and love has been the single most important factor that has led to my success in life. I know that I could not have done this without you. I would also like to thank my sister Beth and brother-in-law Bram, who, being both PhDs, have not only given me support and love, but also very useful advice for surviving graduate school. I would also like to thank my soon-to-be wife Danelle for believing in me at all times, as well as for her incredible love and amazing cooking that has kept me well fed. Most importantly, I would like to thank the Lord God for blessing me with so many opportunities and so many wonderful people in my life.

Contents

List of Figures

List of Tables

List of Abbreviations

1D one-dimensional
2D two-dimensional
3D three-dimensional
ACE average correlation energy
ACH average correlation height
APGD accelerated proximal gradient descent
ASCF adaptive synthetic correlation filter
ASE average squared error
ASEF average of synthetic exact filters
ASM average similarity metric
ATR automatic target recognition
AWGN additive white Gaussian noise
BSCF boosted synthetic correlation filter
CCF constrained correlation filter
CF correlation filter
CM correlation memory
CPU central processing unit
CSSM current slice storage memory
CTE column tail energy
DFT discrete Fourier transform
ECPSDF equal correlation peak synthetic discriminant function
EEMACH eigen extended maximum average correlation height
EER equal error rate
EMACH extended maximum average correlation height
FA/i false alarms per image
FA/s false alarms per second
FDMACE frequency domain minimum average correlation energy

FFT fast Fourier transform
FPGA field-programmable gate array
FRGC facial recognition grand challenge
FSSM future slice storage memory
GMACE Gaussian minimum average correlation energy
GPU graphics processing unit
HOG histogram of oriented gradient
IDFT inverse discrete Fourier transform
IFFT inverse fast Fourier transform
MACE minimum average correlation energy
MACH maximum average correlation height
MCF masked correlation filter
MICE minimum correlation energy
MINACE minimum noise and correlation energy
MMCF maximum margin correlation filter
MOSSE minimum output sum of squared errors
MSE mean squared error
MSESDF minimum squared error synthetic discriminant function
MVSDF minimum variance synthetic discriminant function
OLA overlap-add
OLS overlap-save
ONV output noise variance
OTSDF optimal tradeoff synthetic discriminant function
PACF partial-aliasing correlation filter
PAOLA partial-aliasing overlap-add
PCE peak to correlation energy
PSR peak to sidelobe ratio
RACF reduced-aliasing correlation filter
RATEM reduced-aliasing tail energy minimization
ROC receiver operating characteristic
RTE row tail energy
SDFs synthetic discriminant functions
SIFT scale invariant feature transform
SM storage memory

SNR signal to noise ratio

SVM support vector machine

TDMACE time domain minimum average correlation energy

TE tail energy

TEM tail energy minimization

TEMMACE tail energy minimized minimum average correlation energy

TM total memory

UCF unconstrained correlation filter

UMACE unconstrained minimum average correlation energy

UOTSDF unconstrained optimal tradeoff synthetic discriminant function

VSM video slice memory

ZAASEF zero-aliasing average of synthetic exact filters

ZACCF zero-aliasing constrained correlation filter

ZACF zero-aliasing correlation filter

ZAMACE zero-aliasing minimum average correlation energy

ZAMACH zero-aliasing maximum average correlation height

ZAMMCF zero-aliasing maximum margin correlation filter

ZAMOSSE zero-aliasing minimum output sum of squared errors

ZAOTSDF zero-aliasing optimal tradeoff synthetic discriminant function

ZAUCF zero-aliasing unconstrained correlation filter

ZAUOTSDF zero-aliasing unconstrained optimal tradeoff synthetic discriminant function

Chapter 1

Introduction

Correlation filters (CFs) [1] are well established and useful for a variety of tasks in signal processing and pattern recognition. In these applications, we attempt to extract information from signals of different types. These may be one-dimensional (1D) signals (usually time-varying waveforms); for example, a heartbeat signal, a wireless communication signal, or a speech waveform. More often, the signal is a two-dimensional (2D) image in which we attempt to detect objects or patterns. CFs can also be applied to three-dimensional (3D) signals (videos), to detect a particular type of motion, e.g., action detection. In principle, CFs can be applied to signals of any dimensions, except that the computational and memory complexities grow significantly with increased signal dimensionality.

CFs work in a two step process: training and testing. In the training stage, CFs are designed using a set of training signals. Using these signals, the CF is designed mathematically to minimize some metric, such as the mean squared error between a desired output and the actual output obtained when the training signals are correlated with the CF. Typically, the desired output has a sharp peak at the location of the target, with low values everywhere else. There are many different techniques for designing CFs, which we will review in Chapter 2. Once a filter is designed, it is applied to test data via correlation. Using 1D notation (for simplicity), this correlation operation can be expressed as

$$g(t) = x(t) \otimes h(t)$$
$$= \sum_{t_0} x(t_0) h(t_0 + t) \tag{1.1}$$

where $h(t)$ is the CF template (in the time domain) and $x(t)$ is a test signal. In our notation, $\otimes$ denotes correlation. Because correlation is related to convolution, we can write

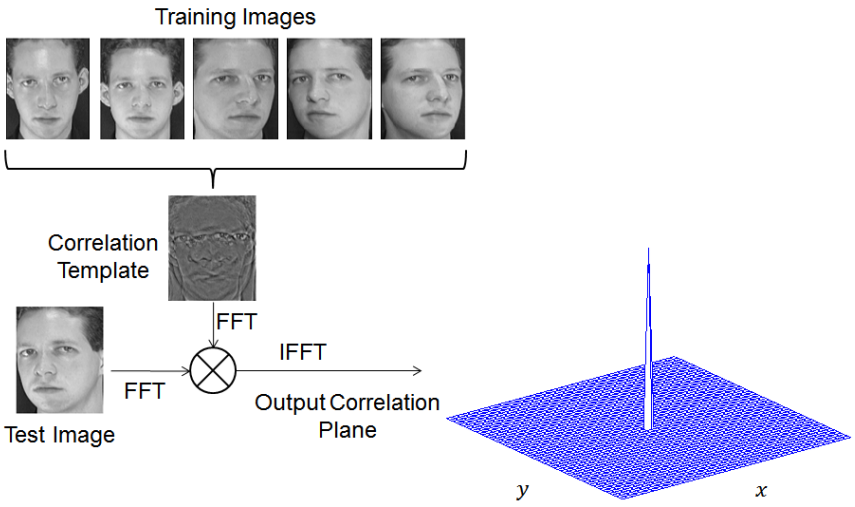
$$g(t) = x(t) \star h(-t) \tag{1.2}$$

where $\star$ denotes convolution. This can be implemented in the frequency domain as

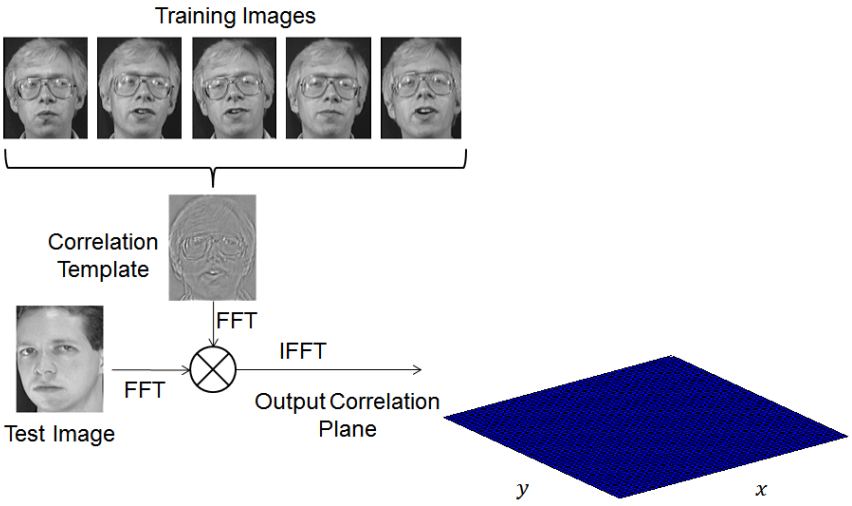
$$g(t) = \mathcal{F}^{-1} \left\{ \mathcal{F}\left(x(t)\right) \odot \mathcal{F}\left(h(-t)\right) \right\} \tag{1.3}$$

where $\mathcal{F}$ and $\mathcal{F}^{-1}$ denote the discrete Fourier transform (DFT) and inverse DFT (IDFT), respectively, and $\odot$ denotes element-wise multiplication. This frequency domain implementation of correlation is much faster than computing the correlation in the time domain because DFTs can be implemented very efficiently using the fast Fourier transform (FFT) algorithm.

CFs are useful for different tasks in signal processing and pattern recognition. One such task is *classification*, which involves assigning a class from a set of predetermined classes to a query signal. For example, one may achieve classification by first determining the aforementioned set of classes. Then, CFs may be trained from example signals of each class. This set of example signals is known as the *training set*. The resulting CFs are then applied to test signals (which should not be in the training set). Ideally, the CF from the correct class will generate a strong output (a *correlation peak*) to the test image, whereas the other CFs will not produce a strong peak in response to the test image. Classification is achieved by assigning the test image to the class whose CF exhibits the strongest response (e.g., the tallest, or the sharpest, peak). We illustrate this process in Fig. 1.1 for the problem of face recognition. In this case, we train two



(a) Strong response to a test image from the same class.



(b) Rejected face image from a different class

Figure 1.1: Schematic of face recognition using CFs.

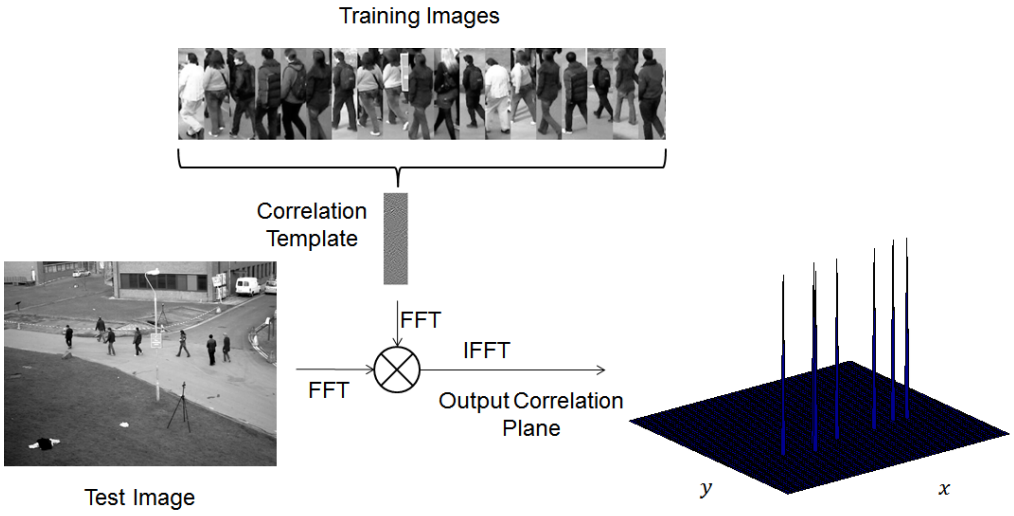


Figure 1.2: Pedestrian localization example using correlation filters. Note that there are seven peaks in the output (one is hard to see), where each peak corresponds to one pedestrian.

CFs, one each for each subject. The CF for subject 1 responds strongly to the test image (which is of subject 1). The CF for subject 2 does not respond strongly to the test image of subject 1. Therefore, the test image is determined to be an image of subject 1.

CFs are also be used for localization. Localization implies finding the location of a specific signal (e.g., an object of some kind) within a larger signal. For example, this could be locating pedestrians in a surveillance video (see Fig. 1.1) or finding the location of a person's eye(s) in a face image.

Recognition refers to simultaneous localization and classification - multiple CFs are applied to an input signal. The outputs from all of the CFs are examined to determine both the location of any possible targets and the corresponding classes. An example of recognition is automatic target recognition (ATR), where a scene is processed to detect the location and class of targets such as tanks, trucks, jeeps, etc.

One major advantage of CFs over other techniques is their property of *shift invariance*. Shift invariance implies that the shifting the input test signal results in an identical shift in the correlation output - in other words, the test signals do not need to be segmented and centered before a

signal is tested. In Fig. 1.2, for example, multiple pedestrians are detected. In the output correlation plane, a correlation peak is present at the location of each pedestrian in the test image. The ability to process unsegmented signals with the target(s) at any location is an important property of CFs.

Finally, another advantage of CFs is a property known as *graceful degradation*. This means that CFs perform well under the presence of occlusion. This is due to the implicit integration in correlation. If a test image is occluded, the CF template will still match up well with the unoccluded parts of each test patch. Therefore, the CF will still respond well to the parts of the image that are visible. This is demonstrated in Fig. 1.3. Note that the occluded test image (in this case, a modified image from the training set) still yields a sharp correlation peak when correlated with the CF. This peak is not as high as the original peak obtained with the full image; however, the fact that it is there in the first place is a major advantage of CFs to other techniques, which cannot handle occlusion as well.

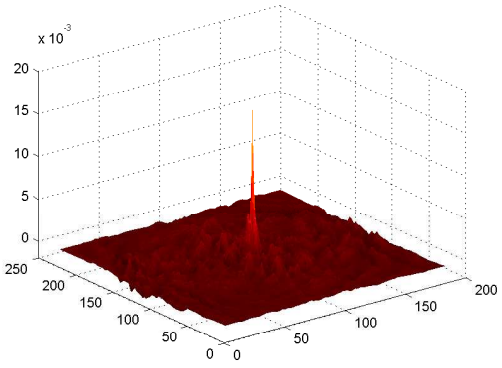
1.1 Thesis Overview

We have illustrated several advantages of CFs that make them useful and attractive tools for pattern recognition. Over the past thirty years, CFs have evolved significantly with the introduction of each CF design (see Chapter 2 for a summary). As CFs have been applied to newer problems, some new research questions have come to the forefront of the field.

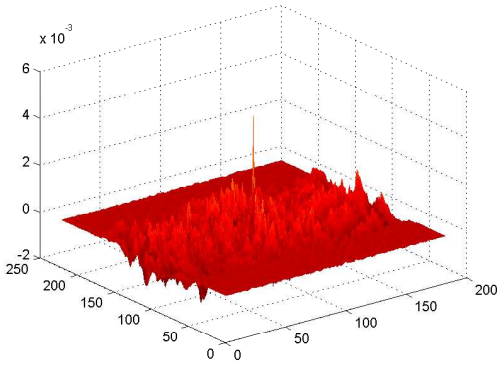
The bulk of CF literature has worked with applications in the image domain. Because correlation is efficiently implemented using the FFT, computational considerations for CF design and application were usually not discussed. However, recent work [2, 3, 4, 5, 6] has investigated the application of CFs to the domain of human action recognition. In this case, the CF is typically a 3D template that is correlated with a 3D test signal (a video) to find the locations of human actions. This results in a 3D output correlation field (see Fig. 1.4 for an example) instead of a 2D correlation peak, as is the case for conventional 2D applications. This move from 2D correlation

(a) An input face image

(b) Occluded face image



(c) Output correlation (no occlusion)



(d) Output correlation (with occlusion)

Figure 1.3: Demonstration of the graceful degradation property of CFs.

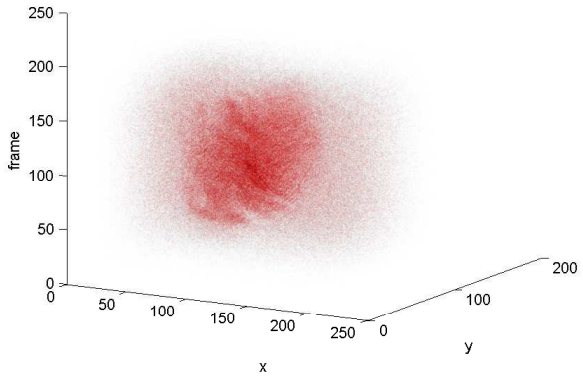
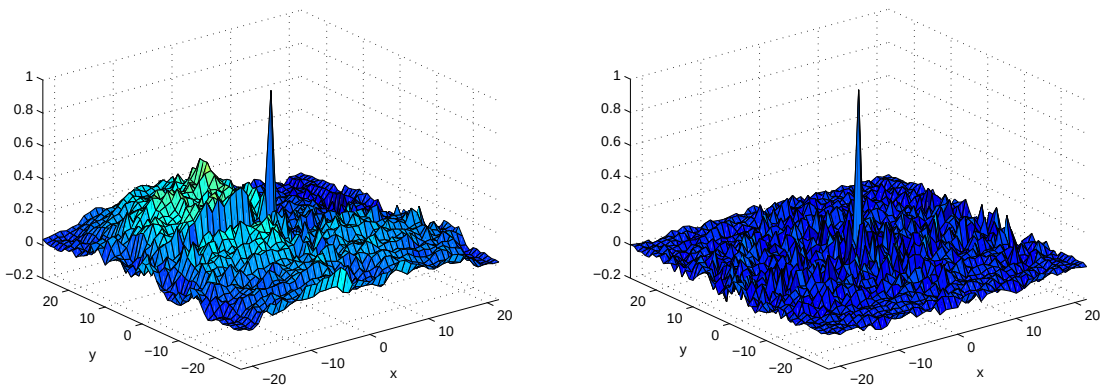


Figure 1.4: Example of a 3D correlation output. Spatial dimensions are given by x and y, and the z axis denotes the frame number. The intensity of the color corresponds to the value of the correlation, with low values being transparent.

to 3D correlation has brought to light various computational issues surrounding CFs. Initially, we identified that the most obvious of these computational issues was the efficient application of the CF itself to test data. Therefore, we investigated the well-known overlap-add (OLA) and overlap-save (OLS) algorithms for multidimensional correlations. These algorithms break the signal up into sections and process each section one at a time. This approach is much faster and more memory efficient than processing the entire signal at once. Through our analysis of OLA and OLS for multidimensional correlations, we uncovered important tradeoffs in each algorithm that have been previously undocumented.

However, applying the CF to test data in an efficient manner is only one computational issue surrounding CFs. Before a CF may be applied to data, it must be designed. Typically, CFs are designed in the frequency domain using DFTs of the training signals. Because the DFT size directly influences the size of the CF template, the efficient application of CFs to test data is directly linked to how the CF is designed in the first place. Because the multiplication of two DFTs in the frequency domain results in circular correlation in the time/space domain, the use of DFTs also introduces aliasing into the output correlation. While this observation is not new, we have found that the analysis of the effects of aliasing is lacking in CF literature, having never been



(a) Conventional Formulation

(b) Our Reformulation

Figure 1.5: Example correlation outputs for the conventional CF design and our design. The ideal output is a sharp peak with low sidelobes. Under the conventional formulation, note that the sidelobes are quite high. With our reformulation of CFs, the output features low sidelobes, which is consistent with the original design criteria.

thoroughly discussed. In investigating these aliasing effects, we have uncovered a significant formulation issue with existing CFs designs in the frequency domain. Because existing designs do not account for aliasing, they are not truly optimal, as their formulations are inherently flawed. This issue has always been known, but has been mostly ignored in the past, as it was assumed that the aliasing issues were minimal. In this thesis, we explore the problem and introduce several solutions that completely eliminate aliasing effects. This results in CFs that are truly optimal and meet their intended design criteria. For example, our reformulation results in correlation outputs that more closely resemble the original design's desired output (see Fig. 1.5). More details will be presented in later chapters.

As mentioned previously, the use of OLA and OLS can dramatically improve the computational requirements of correlating a CF with test data. In this thesis, however, we show that we can make further improvements by reducing the DFT size used for processing each section of the test signal. Doing this introduces errors (from aliasing) into the output correlations. Despite this, we show that is is possible to tolerate this aliasing and still be able to recognize objects. In this case, we design the CFs using the knowledge of the structure of the block filtering scheme,

so that aliasing effects may be minimized. To the best of our knowledge, this is the first time in which CFs have been specifically designed to work in the presence of intentionally introduced aliasing.

As can be seen from the above discussion, these ideas in this thesis challenge conventional notions in CF theory. First, conventional CF theory ***designs*** CFs that are corrupted by aliasing due to circular correlation. This has been a long standing problem that, until now, did not have a solution. We present new designs that eliminate these aliasing effects by designing CFs that correspond to linear correlations. Second, conventional CF theory ***applies*** CFs to test data using linear correlation; we illustrate how CFs can be applied to test data using circular correlations, which intentionally introduces aliasing. Despite this, we are able to eliminate these aliasing effects by carefully designing our CFs to account for and minimize aliasing.

1.2 Thesis Contributions

In this section, we outline the main contributions of this thesis. This thesis can be organized into three major parts, as outlined below.

1.2.1 Designing the CF

(Chapters 2-4)

First, we explore the question, "what is the correct DFT size to use for designing the CF?" Upon initial thought, it is unclear if there is a correct answer to this question. One issue that is initially encountered is that the DFT size directly affects the size of the CF template that is solved for. Intuitively, it would make sense that the template is the same size as the training signals. However, it is also well-known that the multiplication of DFTs in the frequency domain leads to circular correlation, not linear correlation. In the past, this has motivated the practice of zero-padding the training images, as it was widely believed that this would avoid circular

correlation effects. Other than this, the circular correlation issue has been ignored in the past. In this thesis, we delve into this question in detail with the goal of providing more insight and improvement to the filter design process.

Contributions:

- Detailed insight to the DFT size required for conventional CFs to work correctly;
- Illustration of how conventional CF formulations are incorrect;
- A new zero-aliasing correlation filter (ZACF) training framework for eliminating circular correlation effects during training;
- Extension of ZACF to existing CF designs;
- Extensive results of ZACFs on a wide variety of datasets;
- Alternate formulations of the zero-aliasing problem to handle the computational complexity: Tail Energy Minimization, Reduced-Aliasing Correlation Filters, Reduced-Aliasing Tail Energy Minimization, and proximal gradient descent methods.

1.2.2 Applying the CF

(Chapter 5)

After we have designed a CF, we might ask, "what is the most efficient way to apply this CF to test data?" While early work in CFs took advantage of optical Fourier transform techniques, current digital implementations rely heavily on the efficiency of the FFT algorithm. In general, these correlations are not considered to be overly demanding, so their computational aspects are typically overlooked. However, the recent investigation of CFs in 3D has necessitated the use of OLA and OLS, as otherwise these correlations cannot be computed on modern desktop machines.

Contributions:

- An in-depth computational analysis of OLA and OLS for multidimensional correlations, including discussion of parameter selection;
- Discussion of memory requirements regarding OLA and OLS for multidimensional correlations;
- The characterization of the tradeoffs involved between choosing OLA or OLS for filtering a signal;
- The discussion of various design choices when implementing OLA and OLS for multidimensional signals.

1.2.3 Designing the CF with Intentional Aliasing

(Chapter 6)

Traditional CF theory assumes that CFs must be applied to test data using linear correlation. Here, we challenge that assertion, and ask, "can CFs still recognize objects in the presence of aliasing?" In essence, this means that we design the CF to work specifically with the block filtering architecture that will be used to apply it to test data. For example, we may use a generalized OLA architecture that intentionally allows aliasing (from circular correlation) by using short DFT sizes. We show that it is possible to obtain a CF that performs well in the presence of aliasing. The advantage of this alternative block filtering technique is that it requires less computation and memory. We also investigate other ideas, including the design of CFs of varying sizes, sometimes even smaller than the training data.

Contributions:

- Demonstration and analysis of circular correlation effects for a generalized form of OLA, denoted partial-aliasing OLA (PAOLA);
- A new partial-aliasing correlation filter (PACF) training framework that is specifically designed to work with PAOLA and intentional aliasing;

- Demonstration that PACFs outperform conventional CFs in the presence of aliasing;
- Computational and memory analysis for PAOLA;
- Demonstration that it is possible to train a CF that is shorter than the training signals, which in this case reduces aliasing.

1.3 Notation and Conventions

Throughout this document, we will be referring to CFs in both the time domain and frequency domain. In general, we will use the word *template* to refer to the correlation filter in the time (or spatial) domain; we will use the word *filter* to refer to it in the frequency domain. We use the phrases "time domain", "space domain", and "spatial domain" interchangeably, and note that our techniques are not limited to a particular dimension.

In general, our notational convention will denote the size of a D-dimensional signal m as $N_{m,1} \times \cdots \times N_{m,D}$. However, if we are discussing a 1D signal, we will often refer to its length simply as N_m for simplicity. Similarly, we will often be computing DFTs and IDFTs. The size of these transforms are denoted in general by $N_1 \times \cdots \times N_D$. Again, for a 1D signal, we will often drop the subscript and simply refer to the DFT size as N. This notational convention extends to other variables that will be defined for multidimensional algorithms. Additionally, if we are referring to a general mathematical expression that applies to any dimension, we will often use expressions such as $N_{m,d}$ or N_d rather than write out each case for every dimension.

We list the rest of our notation below.

- Matrices will be denoted in capital boldface, e.g., $\mathbf{A}$.
- Vectors will be denoted in lowercase boldface, e.g., $\mathbf{h}$.
- Frequency domain signals will be represented with an overbar, e.g., $\bar{\mathbf{h}}$.
- Time domain signals will have no overbar, e.g., $\mathbf{h}$.
- We denote the transpose with a superscript ^{T}, e.g., $\mathbf{h}^T$.

- We denote complex conjugate with a superscript *, e.g., $\bar{\mathbf{h}}^*$.
- We denote complex conjugate transpose with a superscript $^+$, e.g., $\bar{\mathbf{h}}^+$.
- We denote correlation with the symbol $\otimes$ and convolution with the symbol $\star$.
- If we are referring to specific elements of a signal, image, template, or correlation plane, we will use lowercase non-bold symbols, e.g., $h(n)$ or $h(n,m)$ for time/space domain quantities and capital non-bold symbols, e.g., $H(k)$ or $H(k,l)$ for the frequency domain counterparts.
- We denote image resolutions as $r \times c$, where r is the number of rows (i.e., the number of pixels in the vertical dimension) and c is the number of columns (i.e., the number of pixels in the horizontal dimension).
- The N point DFT and IDFT matrices (of size $N \times N$) are given by $\mathbf{F}_N$ and $\mathbf{F}_N^{-1}$, respectively.
- We indicate an $a \times b$ matrix of all zeros as $\mathbf{0}_{a\times b}$ and an $a \times a$ identity matrix as $\mathbf{I}_a$.

1.4 Organization

This thesis is organized as follows. In Chapter 2, we provide the necessary background on CFs and introduce the details of the aliasing caused by DFT circular correlation. We then present several examples that illustrate in different ways the circular correlation issues in the conventional formulation of CFs. In Chapter 3, we introduce ZACFs as a solution to this problem. We extend ZACFs to a variety of existing CF formulations as well. Because computing ZACFs result in new computational issues, we explore these issues in Chapter 4, presenting several computation and memory efficient alternative formulations. We also present results on a variety of datasets that indicate that our methods perform well and outperform conventional CF designs.

In Chapter 5, we shift our focus to the application of CFs to test data. We start by discussing the use of OLA and OLS for multidimensional correlations and discuss the tradeoffs of each

method with respect to computation and memory. In Chapter 6, we build on the ideas in Chapter 5 by introducing PAOLA and explaining how it is more efficient than OLA at the expense of intentionally introducing aliasing into the correlation outputs. Then, we borrow ideas from ZA-CFs and present PACFs as a new way of designing CFs to work in the presence of aliasing. We summarize our contributions and discuss potential directions for future work in Chapter 7.

Chapter 2

Aliasing Issues in Correlation Filters

Despite the success and use that CFs have enjoyed, there has always been a question about how to properly design them in the frequency domain. Initially, filters were formulated in the frequency domain because computation and optimization in the frequency domain was considerably more efficient than working in the space domain. This is because correlation can be computed in the frequency domain as an element-wise multiplication of two DFTs. It is this fact that has made existing CF designs possible; however, this is also the cause of the circular correlation aliasing effects that are present in existing CF designs.

In this chapter, we demonstrate that there is a serious issue with the formulation of existing CF designs. This problem is very prevalent among many different CF designs. It has been unaccounted for until now. To begin, however, we give a brief introduction to the details of CFs. We then illustrate the issues with current CF formulations in different ways. First, we ask ourselves, "what size DFT should we use to train the CF?" By exploring this question, we expose a serious limitation to existing CF designs. Next, we investigate the formulation of the well-known minimum average correlation energy (MACE) filter, and use it as an illustrative example as to why CF designs are improperly formulated.

2.1 Introduction to CFs

In this section, we give a brief introduction to CFs. The simplest type of CF is a matched filter, in which the template itself is a single signal of interest. For example, for an ATR application, a matched filter could be formed from a single image of a tank. While this matched filter would work well to recognize the target under no distortion, it performs poorly once distortion (i.e. scale, viewpoint, rotation, etc.) is introduced. Therefore, to recognize a single object under any possible distortion, a large number of matched filters are required. This is not practical, especially for applications of recognizing objects with greater space for distortions. For example, human faces can undergo many non-linear transformations, such as expression. Therefore a single matched filter is simply not sufficient.

To remedy this problem, synthetic discriminant functions (SDFs) were introduced. These designs were aimed at determining a single template from a set of training images. This advance signaled the birth of modern CFs. Over the past three decades, the design of CFs has evolved significantly. We can divide all CFs into three categories: constrained CFs, unconstrained CFs, and inequality constrained CFs. Recently, [7] provided a generalized framework for these CFs, denoted the constrained CF (CCF) and unconstrained CF (UCF). In the next chapter, we will take advantage of these formulations when we develop our zero-aliasing formulation, which eliminates aliasing issues due to circular correlation.

2.1.1 Constrained CFs

Constrained CFs place constraints on the filter design such that the response of the CF to each training image is constrained to a particular value. The first of these designs is the equal correlation peak SDF (ECPSDF) [8]. In this case, the template is assumed to be a linear combination of the training images. The dot product of the template and each training image is constrained to a particular value, which corresponds to the value of the correlation output at the origin. This value should represent the correlation peak when the input is a centered training image from the

class of images for which the CF is being designed. However, there is no mechanism to control the sidelobes of the peak; therefore, the resulting peak is not always sharp, and high sidelobes can make it hard to detect.

To solve this issue, one approach was the MACE filter [9]. Like the ECPSDF, the MACE filter imposes peak constraints for each of the training images. However, instead of assuming the template is a linear combination of the training images, the MACE filter minimizes the average correlation energy (ACE) of the output correlation planes with the training images. This essentially reduces the sidelobes of the correlation outputs, which results in much sharper correlation peaks. The MACE filter represents one of the first advanced CFs. Its formulation served as an inspiration for many other filter designs. For example, the minimum correlation energy (MICE) and minimum noise and correlation energy (MINACE) filters [10] adopt a modified method of computing the statistics of the training set for the MACE filter, by placing limits on the energies of particular frequencies in the signal spectrum. This can prevent over-emphasizing particular frequencies. The Gaussian MACE filter (GMACE) [11] and the minimum squared error SDF (MSESDF) [12] generalize the MACE filter by minimizing a mean squared error (MSE) metric between a desired correlation output and the actual correlation output. These filters are generalizations of the MACE filter, because the MACE filter essentially minimizes MSE while assuming the desired output is a delta function.

Another type of constrained CF is the minimum variance SDF (MVSDF) [13]. The MVSDF filter is designed to minimize the output noise variance (ONV) in the output correlation plane. This requires a model of the input noise. The MVSDF is designed to build noise tolerance into CF designs. It was later combined with the MACE filter leading to the optimal tradeoff synthetic discriminant function (OTSDF) filter [14, 15]. The OTSDF filter offers a tradeoff between minimizing ACE and ONV. This tradeoff builds robustness into the CF and allows for better recognition performance. It also serves as one of the most important CFs to date, and inspired many designs of similar optimal tradeoff CFs.

2.1.2 Unconstrained CFs

The introduction of unconstrained CFs was a significant advance to CF theory. In constrained designs, each the dot product of the CF and each training image is constrained to a constant value. However, this does not guarantee that the test images, when correlated with the CF, will produce that constrained peak value. Therefore, unconstrained CFs remove this constraint in filter design. This in general allows for CFs to exhibit better generalization.

The most important unconstrained CF is the maximum average correlation height (MACH) filter [16]. This CF maximizes the correlation height of the output while minimizing ACE and a metric known as average similarity metric (ASM). Minimizing ASM essentially ensures that the correlation outputs from training images from a particular class are as similar as possible. The MACH filter in general performs very well in practice and has remained a popular CF. The MACH filter with only the ACE term (no ASM) is called the unconstrained MACE (UMACE) filter. An optimal tradeoff version of the MACH filter (similar to the OTSDF) was introduced in [17]. This filter is sometimes called the Optimal Tradeoff MACH filter, but we prefer the name unconstrained OTSDF (UOTSDF) filter. Other extensions to the MACH filter include the extended MACH (EMACH) [18], and the eigen EMACH (EEMACH) filter [19].

Recently, the average of synthetic exact filters (ASEF) filter was introduced in [20]. Instead of optimizing a particular metric, ASEF builds one "exact filter" for each training image. Each of these exact filters are then averaged to form the ASEF filter. This method only performs well if many training images (hundreds) are used. The minimum output sum of squared errors (MOSSE) filter [21] aims to remedy this problem by minimizing the error between the desired correlation output and the actual correlation between the template and the training images. The MOSSE filter in general performs better than the ASEF filter. Finally, [22] introduces two new paradigms for training the ASEF and MOSSE filters. First, the boosted synthetic correlation filter (BSCF) modifies the weights of the exact filters for the ASEF formulation using boosting. Boosting is a training method that iteratively updates the weights of a set of weak classifiers that combine to

form a strong classifier. Second, the adaptive synthetic correlation filter (ASCF) introduces an iterative method for retraining the ASEF or MOSSE filter. After the filters are trained initially, they are correlated with the training set and the outputs containing false detections are used to modify the desired correlation outputs and subsequently retrain the filters. This is done many times until the correlation outputs improve.

2.1.3 Inequality Constrained CFs

Another important CF is the maximum margin correlation filter (MMCF) [23]. The MMCF is the first (and only) inequality constrained CF. This filter combines the localization ability of CFs with the generalization ability of support vector machines (SVMs) by maximizing the margin between two classes. Unlike constrained CF designs, which constrain the dot products of the CF and each of the training images to a specific value, the MMCF removes this hard constraint and replaces it with an inequality constraint instead (e.g. the dot product of the CF and each of the training images must be *larger than or equal to* some value). In some cases, a slack variable allows some dot products to fall on the wrong side of the margin (this is allowed in order to get a more generalized margin).

2.1.4 Applications of CFs

There are a wide variety of applications for CFs in biometrics, automatic target recognition, landmark detection, and action recognition. For example, CFs are used for face recognition in [24]. More recently, [25] demonstrated the use of CFs for occluded face recognition. This demonstrated yet another advantage of CFs, namely their graceful degradation in the presence of occlusions. A deformable pattern matching method is coupled with CFs in [26] to recognize human iris images. More recently, ocular recognition has become of greater interest to the biometric community, and CFs have been applied to this problem in [27]. CFs have also been suggested for biometric key binding in [28]. Phase-only correlation methods, which are similar

to CFs, have been used to correlate the phase information in two images. Biometric applications of these techniques include iris recognition [29], as well as face, fingerprint, palmprint, and finger-knuckle recognition [30, 31].

CFs have been used for a long time for automatic target recognition [23, 32] and to track moving objects in video [21, 32, 33]. Pedestrian detection has been demonstrated in [34]. CFs have been also used as interest point detectors. For example, [20, 23] uses CFs for eye localization in images of human faces, and [35] uses CFs for interest point detection in object alignment. CFs have also proven useful for detecting and locating text strings for document processing [36].

CFs have also been extended to three dimensions (two spatial dimensions and one temporal dimension) and have been applied to detect and classify human actions [2, 3, 4, 5, 6]. In [4], a motion model is combined with 2D CFs to classify human activities. The use of CFs is not limited to pixels; they have been applied to work with feature vectors such as optical flow [2], histogram of oriented gradient (HOG), or scale invariant feature transform (SIFT) features [35, 37].

2.1.5 Aliasing and CFs

Despite all the work in CFs, the circular correlation problem (which we will explain in this chapter) has been mostly ignored. This problem was initially observed in [38], which proposed reformulating the MACE filter [9] in the space domain to avoid circular correlation. However, computation of the MACE filter in the space domain is of significant complexity. Furthermore, there was never a proposed frequency domain solution. More recently, [39] explored several methods to reduce circular correlation effects. These methods use zero padding and/or windowed training images to lessen the effects of circular correlation. One of these methods, originally mentioned in [20], multiplies the training images by a cosine window to reduce edge effects (this approach is explored in greater detail in [39]). Windowing is undesirable, however, because it fundamentally changes the content of the training images. All of the methods discussed in [39] fall short, as they do not adequately reduce circular correlation effects. Outside of the field of

CFs, convolutional sparse coding was recently extended to the frequency domain [40]. In this application, filters are learned in the frequency domain. Although the authors admit that circular convolution is present, they only offer heuristics to fix these effects, such as utilizing symmetric convolution.

We introduced zero-aliasing correlation filters (ZACFs) for the MACE filter in [41]. A main contribution of this thesis is to show how this method solves the circular correlation problem in CFs. The details of our approach will be explained in the next chapter.

2.2 Introduction to Circular Correlation

Before we begin discuss CF aliasing issues in detail, we provide a brief discussion of circular correlation.

Suppose we have two signals, $x(n)$ and $y(n)$. One method to compute the correlation is to do so in the time domain using a sliding correlator. However, it is also possible to compute the correlation in the frequency domain. This is because the multiplication of two DFTs in the frequency domain corresponds to a circular correlation in the time domain [42]. This is a well known fact, but oftentimes circular correlation is assumed to be approximately the same as linear correlation. Indeed, this is the assumption that was used for the bulk of CF theory. Despite this, DFTs may still be used to compute linear correlation. In order for this to work, both signals must be sufficiently zero padded prior to taking the DFT. If signal $x(n)$ is of length N_x and signal $y(n)$ is of length N_y, then both signals must be padded with zeros to at least size $N_x + N_y - 1$. The resulting correlation is still technically circular, but it is the same as a linear correlation, as the zero-padding essentially nulls out any potential aliasing.

To illustrate the process of computing correlation in the frequency domain, we present an example. First, we show a signal $x(n)$ of length $N_x = 16$ in Fig. 2.1a and signal $y(n)$ of length $N_y = 7$ in Fig. 2.1b. First, we illustrate linear correlation in Fig. 2.1c through 2.1e. Here, we use a DFT of size $N_{lin} = N_x + N_y - 1 = 22$. In Fig. 2.1c, we show $x(n)$ with $N_y - 1 = 6$ zeros

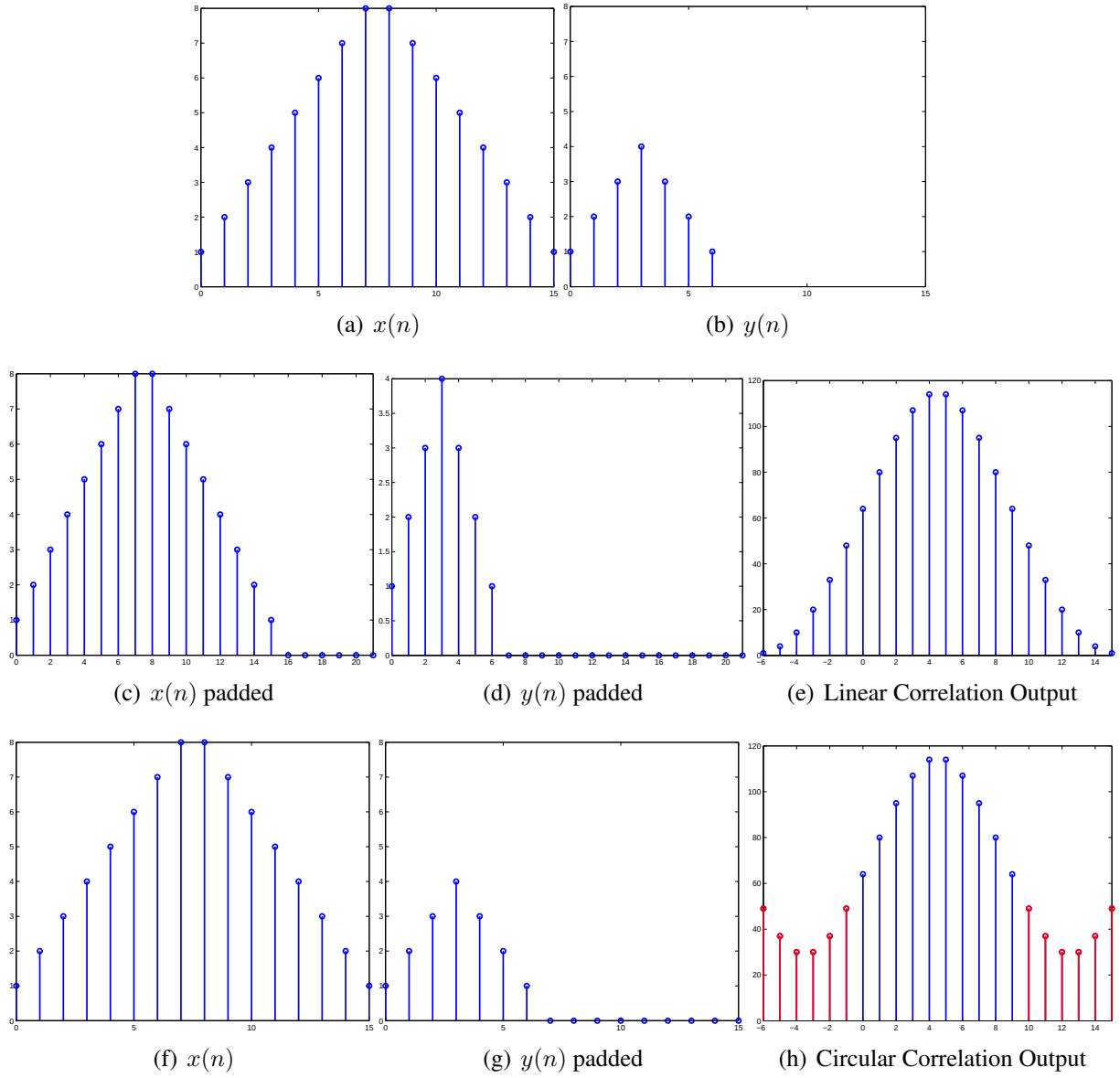
appended. In Fig 2.1d, we show $y(n)$ with $N_x - 1 = 15$ zeros appended. We then flip the padded $y(n)$ (which is necessary to compute correlation, not convolution) and take the N_{lin}-point DFTs of both signals. We element-wise multiply them and take the N_{lin}-point IDFT to return to the time domain. This output is the linear correlation shown in Fig. 2.1e. It is identical to a sliding window correlation output.

Next, we illustrate circular correlation in Fig. 2.1f through 2.1h. Here, we use a DFT of size $N_{circ} = 16$. In Fig. 2.1f, we show $x(n)$ with no zeros appended (because $N_x = N_{circ}$). In Fig 2.1g, we show $y(n)$ with $N_{circ} - N_y = 9$ zeros appended. We then flip the padded $y(n)$ and take the N_{circ}-point DFT of both signals. We element-wise multiply them and take the N_{circ}-point IDFT to return to the time domain. This circular correlation output is of length N_{circ}. However, we cyclically extend it so that it is of length N_{lin}, and plot it in Fig. 2.1h. Note that the red values on each tail represent values that are corrupted from circular correlation. There are exactly $N_{lin} - N_{circ}$ of these values on each tail.

Note that if $x(n)$ and $y(n)$ are much different in size, there will be less severe circular correlation effects, using the methods we outlined here. As the signals become closer in size, aliasing increases. In fact, if the signals are the same length, all of the circular correlation output values will be different from the linear correlation output except for the value at the origin! This is illustrated in Fig. 2.2. This is actually the case for conventional CF designs (when the DFT size is the same as the image size), as the designed CF template will then be the same size as the training images.

2.3 Investigation of DFT Size for Filter Design

When designing a CF in the frequency domain, the first question that comes to mind is, "what DFT size should be used to train the filter?" Typically, this issue is not discussed much in literature, as it is viewed as an implementation detail. Typically, CFs are trained with a set of cropped training signals (or images) that serve as a representative sample of a particular class.

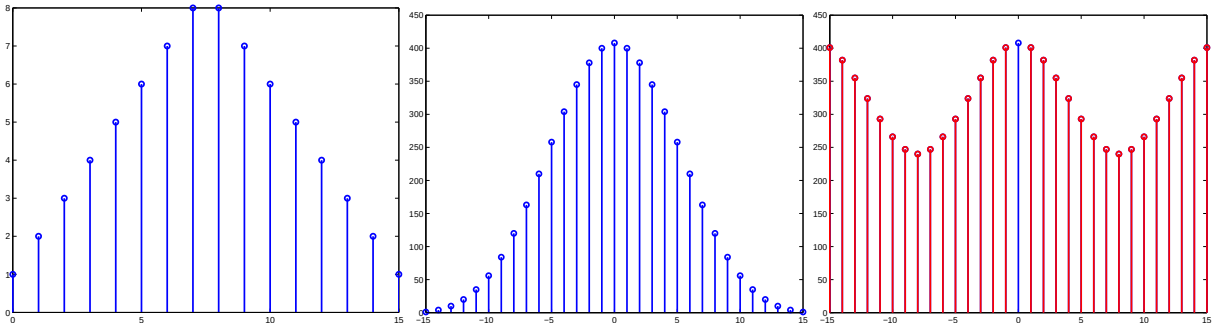


(a) $x(n)$ (b) $y(n)$

(c) $x(n)$ padded (d) $y(n)$ padded (e) Linear Correlation Output

(f) $x(n)$ (g) $y(n)$ padded (h) Circular Correlation Output

Figure 2.1: Example illustrating the difference between linear and circular correlation. See text for details.

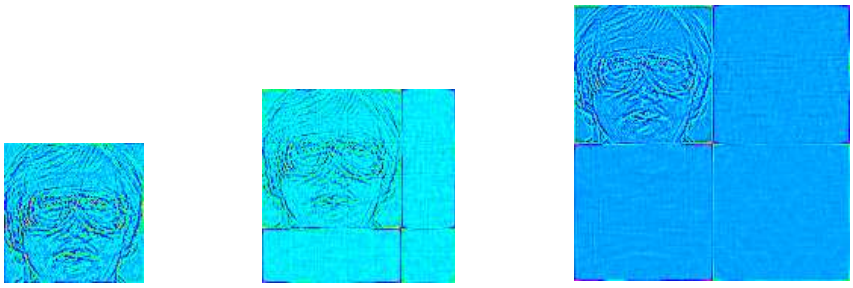


(a) Input Signals, $x(n) = y(n)$ (b) Linear Correlation Output (c) Circular Correlation Output

Figure 2.2: Example illustrating the difference between linear and circular correlation with equal signal lengths. The red values in the circular correlation output denote values that have been corrupted due to aliasing.

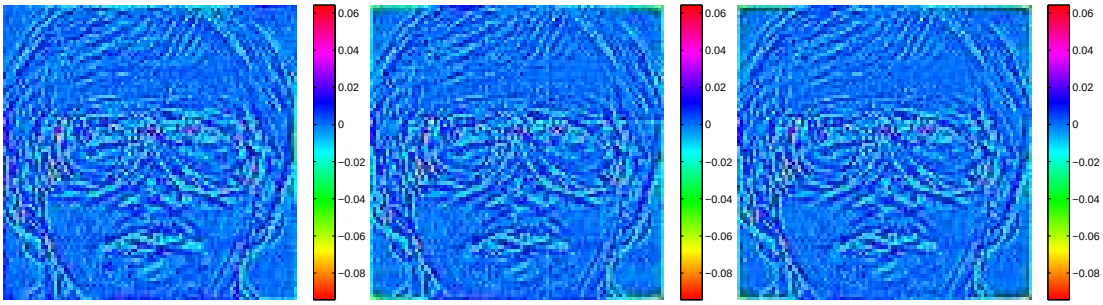
Therefore, the immediate logical answer to this question is to simply use a DFT size equal to the image size. Another answer is to use a DFT that is larger than the image, but that is a power of 2, for efficiency. Still another answer is to use a DFT size that is at least of size $N_d = 2N_{x,d} - 1$, where $N_{x,d}$ is the size of the training signal in dimension d. The motivation for this choice is that this provides enough zero-padding to avoid circular correlation. This is because if we are correlating two signals of size $N_{x,d}$, we must zero-pad each to size $2N_{x,d} - 1$ or greater prior to taking the DFT. Then, the multiplication of the two signals in the frequency domain will correspond to a linear correlation, rather than a circular correlation, in the space domain. This argument makes the assumption that the CF template that we are solving for is also of length $N_{x,d}$, or can at least be assumed to be of this length.

However, there is a problem! When we design CFs using different DFT sizes, we get different templates! This is illustrated in Fig. 2.3. In this example, we train a CF using three face images of size 92×92. We use three different DFT sizes: 92×92, 128×128, and 183×183. In each case, we obtain a template of a different size, equal in size to that of the DFT that was used. Typically, we would crop the template to size 92×92. This is shown in Fig. 2.4. Note that even in this case, the three templates are different from each other. In particular, they all exhibit different aliasing effects that are caused by the DFT size that was used to train them.



(a) DFT: 92×92 (b) DFT: 128×128 (c) DFT: 183×183

Figure 2.3: The designed templates using different DFT sizes.



(a) DFT: 92×92 (b) DFT: 128×128 (c) DFT: 183×183

Figure 2.4: Cropped templates using different DFT sizes. Note that the edges of the templates in particular are different. Also, observe the vertical and horizontal lines present in the 128×128 template.

Figure 2.5: Faces used in the experiment. All have been cropped to size 92×92.

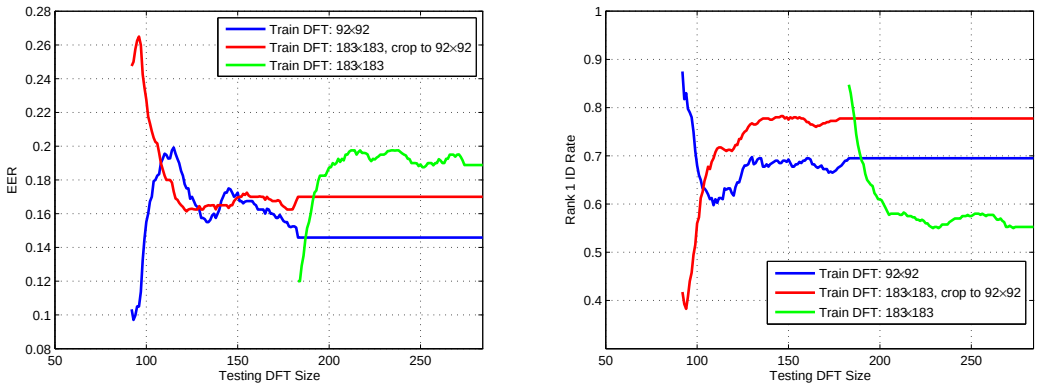
The question still remains, which template is the best? To further illustrate this point, we perform an experiment. We use the AT&T/ORL Face Dataset [43]. In this dataset, there are 40 subjects, with 10 images of each subject. Each image is 112×92. To simplify this experiment, we crop each image to size 92×92 by removing 10 pixels from the top and bottom of each image. See Fig. 2.5 for example images for each subject.

We train UOTSDF filters using 9 training images from each class. The filters are trained in three different ways:

1. DFT of size 92×92;
2. DFT of size 183×183, and then crop template to size 92×92;
3. DFT of size 183×183.

These filters were then tested on the remaining image for all subjects. This resulted in a total of 40 test images (one from each class) that were tested with 40 filters each, for a total of 1600 correlations per cross validation step. This process was repeated 10 times to capture every possible training set.

For each correlation output, we calculate peak to correlation energy (PCE) as described in [1]. PCE is a measure of peak sharpness and is a good metric that may be used for classification.



(a) EER as a function of test DFT size

(b) Rank-1 ID as a function of test DFT size

Figure 2.6: Performance of CFs while varying the DFT size used at testing.

PCE is computed as

$$PCE = \frac{|c_{peak}|^2}{\sum_i \sum_j |c_{i,j}|^2} \tag{2.1}$$

where c_{peak} is the value of the correlation peak, and $c_{i,j}$ represents the correlation plane. Using the PCE scores for each filter, we varied the threshold to generate receiver operating characteristic (ROC) curves, yielding the system's true positive rate and false positive rate for a given threshold. To consolidate these curves into a single score, we compute the equal error rate (EER). We also perform classification on each of the test images and compute the rank-1 ID rate.

We run the above experiment multiple times, varying the DFT size used to compute the correlation in the *testing* stage while keeping the trained templates the same. For the 92×92 templates, we start with a DFT of size 92×92. Since the test images are also 92×92, this is a circular correlation. The correlation continues to be a circular correlation until the test DFT is larger than or equal to 183×183. Similarly, for the 183×183 template, we start with a DFT of size 183×183. Since the test images are 92×92, this is a circular correlation. The correlation continues to be a circular correlation until the test DFT is larger than or equal to 274×274.

We plot the results of this evaluation in terms of EER in Fig. 2.6a and in terms of Rank-1 ID rate in Fig. 2.6b. We observe that:

- For method 1 (train DFT size 92×92) and method 3 (train DFT size 183×183), performance is best with the test DFT size is 92×92 and 183×183, respectively. Both of these points represent *circular* correlation.
- For both methods 1 and 3, performance decreases as DFT size increases.
- Once the DFT size is greater than or equal to 183×183 (method 1) or 274×274 (method 3), the correlation becomes linear and the performance is constant.
- For method 2 (template size 92×92) we notice the opposite: performance is poor for a DFT size 92×92 (circular correlation). As the DFT size increases, aliasing due to circular correlation decreases; performance increases, and becomes constant once the correlation is linear (DFT size larger or equal to 183×183).

This experiment illustrates several important points. First, conventional CF designs are very dependent on the DFT size that is used in both training and testing. If these two DFT sizes are not the same, then performance is poor. This can be a very serious problem if the training images are not the same size as the testing images, which is often the case. Furthermore, we have observed that performance is the best when the test DFT is a *circular* correlation. While this result may not seem immediately intuitive, it will become more clear in Section 2.4, where we investigate CF formulations in more detail. The answer, of course, is that existing CF designs are inherently (and unintentionally) formulated as circular correlations. Therefore, they only "work" if the circular correlation is the same during testing as it was in training (i.e., the DFT size must be the same).

Method 2 represents a typical implementation of CFs in which the training DFT template is cropped from a larger template. This is done because the template intuitively should be the same size as the training images. Furthermore, this implementation is necessary for localization tasks, where the test images are much larger than the training images (and the goal is to find the precise location of the target within the scene, rather than to simply perform classification). We have also consistently observed that this method typically performs better than the other methods

assuming the filter is applied to data with a *linear* correlation. Therefore, method 2 is typically our baseline method.

2.4 Investigation of CF Formulations

In this section, we discuss issues regarding the design of CFs in the frequency domain, namely the unintended effects of circular correlation. This problem was initially observed in [38], which proposed reformulating the popular MACE filter [9] in the space domain rather than the frequency domain to avoid circular correlation. However, computation of the MACE filter in the space domain is of significant computational complexity. More recently, [39] explored several methods to reduce circular correlation effects. These methods use zero padding and/or windowed training images to lessen the effects of circular correlation. One of these methods, originally mentioned in [20], multiplies the training images by a cosine window to reduce edge effects (this approach is explored in greater detail in [39]). Windowing is undesirable, however, because it fundamentally changes the content of the training images. All of the methods discussed in [39] fall short, as they do not adequately reduce circular correlation effects.

To illustrate the circular correlation problem, we introduce two different formulations of the MACE filter - one in the frequency domain (FDMACE) and one in the time domain (TDMACE). Then, we will illustrate that FDMACE is not as optimal as TDMACE.

2.4.1 FDMACE

The MACE filter is designed to respond well to training signals (in fact, the correlation output for positive training signals is typically constrained to be exactly 1 at the origin). In addition, this filter attempts to minimize the ACE of the entire correlation plane. The result of this minimization, coupled with the peak constraints, is a sharp correlation peak with low sidelobes.

Let us consider the 1D MACE filter. Let us assume that we will be training the filter using

training signals, $x_q(n)$, with $q = 1, \ldots, Q$ and $n = 0, \ldots, N_x - 1$. We will use $H(k)$ and $X_q(k)$ to denote the CF and the training signals, respectively, in the frequency domain. The ACE for these training signals is given by

$$\text{ACE} = \frac{1}{Q} \sum_{q=1}^{Q} \sum_{n=0}^{N_x-1} |c_q(n)|^2 \tag{2.2}$$

where $c_q(n)$ is the correlation output of the filter and the qth signal. This can be expressed in the frequency domain with Parseval's theorem,

$$\text{ACE} = \frac{1}{NQ} \sum_{q=1}^{Q} \sum_{k=0}^{N-1} |C_q(k)|^2 \tag{2.3}$$

where $C_q(k)$ is the N-point DFT of $c_q(n)$. In the traditional MACE formulation, $C_q(k) = H(k)X_q^*(k)$, and ACE may be expressed as

$$\text{ACE} = \frac{1}{NQ} \sum_{q=1}^{Q} \sum_{k=0}^{N-1} |X_q(k)|^2 |H(k)|^2 \tag{2.4}$$

If we represent $H(k)$ as a vector $\bar{\mathbf{h}}$ and $X_q(k)$ as a diagonal matrix $\bar{\mathbf{X}}_q$ with the elements of $X_q(k)$ along the diagonal, ACE may be expressed in matrix-vector form as

$$\begin{aligned} \text{ACE} &= \frac{1}{NQ} \sum_{q=1}^{Q} \bar{\mathbf{h}}^+ \bar{\mathbf{X}}_q \bar{\mathbf{X}}_q^* \bar{\mathbf{h}} \\ &= \bar{\mathbf{h}}^+ \left[\frac{1}{NQ} \sum_{q=1}^{Q} \bar{\mathbf{X}}_q \bar{\mathbf{X}}_q^* \right] \bar{\mathbf{h}} \qquad (2.5) \\ &= \bar{\mathbf{h}}^+ \mathbf{D} \bar{\mathbf{h}} \qquad (2.6) \end{aligned}$$

where

$$\mathbf{D} = \frac{1}{NQ} \sum_{q=1}^{Q} \bar{\mathbf{X}}_q \bar{\mathbf{X}}_q^* \tag{2.7}$$

is an $N \times N$ matrix.

The standard MACE formulation will minimize Eq. 2.6 subject to the constraints

$$\bar{\mathbf{X}}^+ \bar{\mathbf{h}} = N\mathbf{u} \tag{2.8}$$

In this case, $\bar{\mathbf{X}}$ (no subscripts) is a matrix whose columns contain the N-point DFTs of the training signals, and $\mathbf{u}$ is a ground truth vector, whose elements correspond to each training signal. Typically, 1 is used for a positive class training element and 0 is used for a negative training element. The solution for the MACE filter is given by [9]

$$\bar{\mathbf{h}} = N\mathbf{D}^{-1}\bar{\mathbf{X}} \left(\bar{\mathbf{X}}^+ \mathbf{D}^{-1} \bar{\mathbf{X}}\right)^{-1} \mathbf{u} \tag{2.9}$$

As we mentioned previously, the implementation of Eq. 2.9 is ambiguous. Specifically, the DFT size, N, must be chosen. As shown previously, different choices of N will yield different filters. Therefore, we can remove this choice by formulating the MACE filter in the time domain instead. We can then compare FDMACE to the TDMACE formulation.

2.4.2 TDMACE

In general, computing CFs in the time or space domain is not as attractive due to computational concerns. In this section, we formulate this problem for discrete 1D signals. Note that a generalized space domain implementation of MACE can also be found in [38].

Consider a CF $h(n)$ and a set of input signals $x_q(n)$, $q = 1, \ldots, Q$. For the purpose of this example, $h(n)$ is defined for $0 \leq n \leq N_h - 1$ and is 0 elsewhere; similarly, each $x_q(n)$ is defined for $0 \leq n \leq N_x - 1$ and is 0 elsewhere. We assume that the template length N_h is shorter than the training signal length N_x. In most practical cases $N_h = N_x$. Correlation in the time domain may be defined as

$$c_q(n) = \sum_{k=0}^{N_h-1} h(k)x_q(k-n) \tag{2.10}$$

Here, $c_q(n)$ will be zero outside of the range $-(N_x-1) \leq n \leq N_h-1$. This corresponds to $N_h + N_x - 1$ correlation values. We can define the correlation energy for $x_q(n)$ as

$$\begin{aligned} E_{c,q} &= \sum_{n=-(N_x-1)}^{N_h-1} |c_q(n)|^2 \\ &= \sum_{n=-(N_x-1)}^{N_h-1} \left| \sum_{k=0}^{N_h-1} h(k)x_q(k-n) \right|^2 \\ &= \sum_{n=-(N_x-1)}^{N_h-1} \left(\sum_{k=0}^{N_h-1} h(k)x_q(k-n) \right) \left(\sum_{l=0}^{N_h-1} h(l)x_q(l-n) \right)^* \qquad (2.11) \\ &= \sum_{n=-(N_x-1)}^{N_h-1} \sum_{k=0}^{N_h-1} \sum_{l=0}^{N_h-1} h(k)x_q(k-n)x_q^*(l-n)h^*(l) \qquad (2.12) \end{aligned}$$

Rearranging the summations, we can express this as

$$\begin{aligned} E_{c,q} &= \sum_{k=0}^{N_h-1} \sum_{l=0}^{N_h-1} h(k) \left(\sum_{n=-(N_x-1)}^{N_h-1} x_q(k-n)x_q^*(l-n) \right) h^*(l) \\ &= \sum_{k=0}^{N_h-1} \sum_{l=0}^{N_h-1} h(k)R_q(k,l)h^*(l) \qquad (2.13) \end{aligned}$$

where

$$R_q(k,l) = \sum_{n=-(N_x-1)}^{N_h-1} x_q(k-n)x_q^*(l-n) \tag{2.14}$$

We can rewrite Eq. 2.13 as

$$E_{c,q} = \mathbf{h}^{+}\mathbf{R_q}\mathbf{h} \tag{2.15}$$

where $\mathbf{h}$ is an $N_h \times 1$ vector given by

$$\mathbf{h} = \begin{bmatrix} h(0) \\ \vdots \\ h(N_h - 1) \end{bmatrix} \tag{2.16}$$

and matrix $\mathbf{R}_q$ is an $N_h \times N_h$ matrix, in which the element for row k and column l is given by Eq. 2.14. The matrix $\mathbf{R}_q$ is Toeplitz and conjugate symmetric (see Appendix A).

We may now compute the average correlation energy as

$$\begin{aligned} \bar{E}_c &= \frac{1}{Q}\sum_{q=1}^{Q} E_{c,q} \\ &= \frac{1}{Q}\sum_{q=1}^{Q} \mathbf{h}^{+}\mathbf{R_q}\mathbf{h} \\ \bar{E}_c &= \mathbf{h}^{+}\left(\frac{1}{Q}\sum_{q=1}^{Q}\mathbf{R_q}\right)\mathbf{h} \\ &= \mathbf{h}^{+}\bar{\mathbf{R}}\mathbf{h} \end{aligned} \tag{2.17}$$

where

$$\bar{\mathbf{R}} = \frac{1}{Q}\sum_{q=1}^{Q}\mathbf{R_q} \tag{2.18}$$

is also a Toeplitz conjugate symmetric matrix. To formulate the TDMACE filter, we want to minimize $\bar{E}_c$ subject to several constraints. In particular, we want the filter to generate a peak at the origin of each of the positive-class training signals. This correlation output at the origin is given by

$$c_q(0) = \sum_{k=0}^{N_h-1} h(k)x_q(k)$$
$$= \mathbf{h}^T \mathbf{x}_{q,N_h} \tag{2.19}$$

where

$$\mathbf{x}_{q,N_h} = \begin{bmatrix} x_q(0) \\ \vdots \\ x_q(N_h-1) \end{bmatrix} \tag{2.20}$$

is the vector formed by the first N_h values of x_q. We can formulate the TDMACE filter design by minimizing $\bar{E}_c$ subject to the linear constraints $\mathbf{X}^T\mathbf{h} = \mathbf{u}$. Here, the columns of the $N_h \times Q$ matrix $\mathbf{X}$ are the vectors $\mathbf{x}_{q,N_h}$. The $Q \times 1$ constraint vector $\mathbf{u}$ is chosen based on the training signals. Typically, a 1 is used for a true class signal and a 0 is used for a negative class signal. Note that this quadratic minimization problem subject to linear constraints is very similar to the FDMACE filter formulation. The optimal TDMACE CF template is given by

$$\mathbf{h} = \bar{\mathbf{R}}^{-1}\mathbf{X}\left(\mathbf{X}^T\bar{\mathbf{R}}^{-1}\mathbf{X}\right)^{-1}\mathbf{u} \tag{2.21}$$

2.4.3 Comparing FDMACE to TDMACE

Let us consider a simple 1D example to see how the TDMACE compares to the FDMACE. To illustrate both formulations, we use data from the MIT-BIH Arrhythmia Database [44]. In this case, we have extracted the first 5 minutes of record 103 in the database, resulting in 355 heartbeat cycles. For reference, we have plotted all 355 segmented heartbeat signals in Fig. 2.7. Each signal is 301 samples long and represents a single heartbeat, which has been segmented by a cardiologist expert based on the location of the main peak. The purpose of using these signals

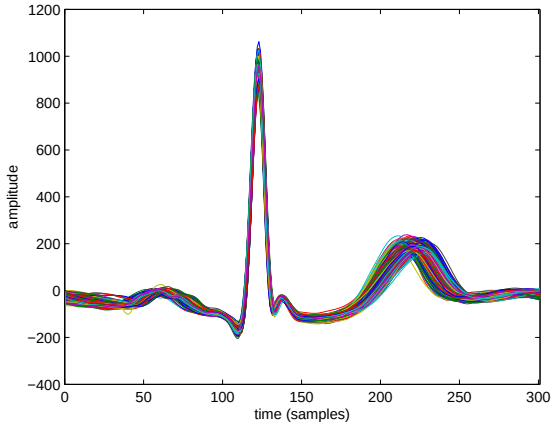
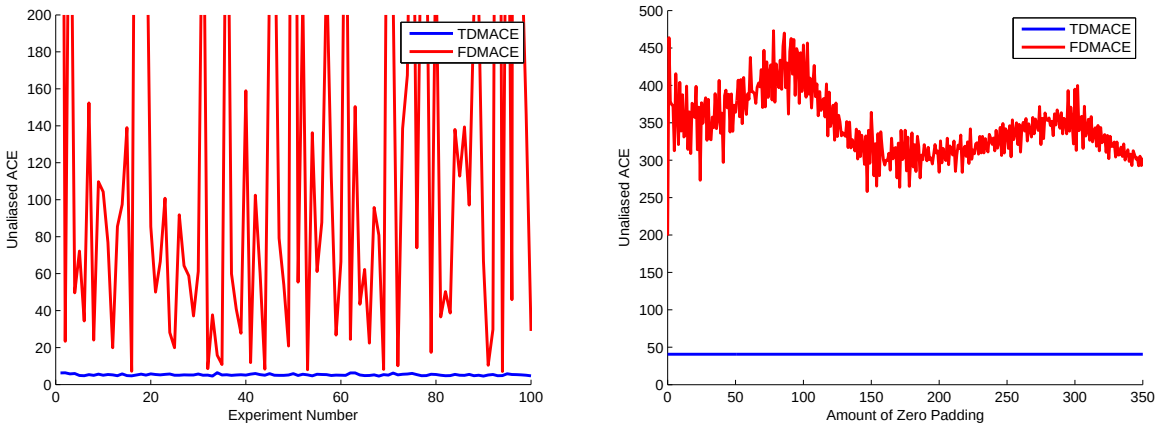


Figure 2.7: Plot of the segmented MIT-BIH heartbeats used as a test to evaluate TDMACE and FDMACE filters.

is to use real signals to demonstrate CF performance. We do not imply that CFs should be used for heart beat detection or Arrhythmia classification.

To begin with, we we choose our correlation template to be $N_h = N_x$ samples long for the TDMACE formulation. For the FDMACE formulation, we use $N = N_x$ for the DFT size. In our first experiment, we randomly select $Q = 10$ heartbeat signals from the dataset and build both a TDMACE and FDMACE correlation template. We apply the resulting correlation template to the training examples (using time domain correlation) and compute the *unaliased ACE*. We use the term unaliased ACE to make a distinction from ACE, which is given in Eq. 2.6. Unaliased ACE is simply the average correlation energy of the *linear* correlations of the training samples and the template. Ideally, the answers should be identical, if the TDMACE and FDMACE are identical. However, what we find is actually the opposite - the unaliased ACE achieved by the conventional MACE formulation (FDMACE) is always higher than the unaliased ACE achieved by the TDMACE formulation. We have repeated this experiment 100 times and show the results in Fig. 2.8a. This same result has been observed on other signals as well, and indicates that there is an issue with the FDMACE formulation, as it is clearly not optimal.

After making this observation, we determined that the reason for the FDMACE's poor performance was the formulation of ACE in the frequency domain (initially, Eq. 2.4). This expression



(a) (b)

Figure 2.8: Unaliased ACE for FDMACE and TDMACE. (a) TDMACE achieves a lower unaliased ACE than FDMACE, regardless of the training set. (b) Adjusting the zero padding to generate a new FDMACE filter. Here, the filter size varies as a function of zero padding.

features the multiplication of the filter with the DFT of each training signal. Because we are using the DFT, this is inherently a circular correlation! One common misconception is that zero padding the training signals to a size $N \geq 2N_x - 1$ prior to taking the DFT will avoid this circular correlation. However, because the DFT size (N) is now longer than the original training signals, the resulting template will be N samples long. Note that this in itself is another issue, as we really desire a template size that is the same size as the training signals. Nevertheless, we demonstrate this approach in Fig. 2.8b. Here, we have taken the same training set ($Q = 10$) and have varied the zero padding used, denoting the number of zeros appended to each signal as p. Note that the length of the DFT and the resulting template is now $N = N_x + p$. Because the template size varies, we standardize the unaliased ACE computation by normalizing each template to unit energy and computing normalized correlation (i.e. each signal segment corresponding to each correlation lag is also normalized to unit energy). If this method is to work, the unaliased ACE should decrease to a minimum at a zero padding of 300 and above (because $N_x = N_h = 301$, then $N_x + N_h - 1 = 601$, which means the necessary zero padding to avoid circular correlation must be at least $p = 300$). However, this is not the case! The unaliased ACE does not improve

as p increases. This is because, for the ACE expression to represent linear correlation, both the training signals *and* the correlation template must be zero padded! For example, if we pad the training signals with $p = 300$ zeros, we use DFTs of size 601, and therefore obtain a correlation template of size 601. This template is *nonzero* for all 601 entries. Thus, it is the template we solve for, not the training signals that we use, that contributes to the circular correlation effects in this case. If we were to avoid the effects of circular correlation, we would have to pad the training images *and* force the template that we solve for to have zeros in the last 300 positions. To accomplish this in the frequency domain, we reformulate the FDMACE in the next chapter. We call this enhanced filter design a zero-aliasing correlation filter (ZACF).

2.5 Conclusions and Discussion

In this chapter, we presented two different viewpoints of the current issues with conventional CF designs. First, we examined the effect of DFT size on CF design. Next, we illustrated how current CF designs are incorrectly formulated, as they (unintentionally) assume a circular correlation. We summarize the main points of this chapter below.

- Conventional CF formulations give different templates for different DFT sizes.
- Conventional designs assume a circular correlation at training time, regardless of how much zero-padding is used on training images.
- Performance for conventional designs is best when the testing DFT is the same size as the training DFT; however, if this is the case, the testing process is in fact a circular correlation. This is, itself, an important observation that has not been made previously.
- Conventional CF designs perform worse for linear correlation than they do with circular correlation. This is an important observation, and can be used to reduce computation for classification tasks.
- Because the DFT size at testing must be the same as the DFT size at testing, conventional

CF designs inherently are not well designed for localization tasks, i.e. when the test images are different in size from the training images.

- The TDMACE formulation achieves a lower unaliased ACE on the training set than the conventional FDMACE formulation. This is an indication that the FDMACE formulation is in fact, not optimal *if linear correlation is assumed at testing time*, which is the case in CF theory. This type of problem is found in the MACE filter as well as many other CF designs, as we will show in the next chapter.
- The problem with current CF designs is that they are nonzero for all elements of the template. This is the precise cause of circular correlation aliasing during training. This observation leads to our enhanced CF formulation, ZACFs, which we present in the next chapter.

Chapter 3

Zero-Aliasing Correlation Filters

In the previous chapter, we introduced CFs, discussed circular correlation, and illustrated the formulation problem for existing CF designs. This formulation problem is that CFs implicitly assume circular correlation when the CF is trained. This means that good performance is only attained at testing time if the DFT at testing time is the same size as the DFT at training time (i.e., a circular correlation). This may be possible for very simple tasks (such as when the training images are the same size as the testing images), but it is not a safe assumption for many other tasks, in particular localization. Furthermore, this subtlety has not been common knowledge, because the application of CFs has has historically been implemented using linear correlation, not circular correlation. Therefore, if linear correlation is to be used at testing time, the optimization function for CF designs ought to express correlation as a linear correlation as well. In this chapter, we introduce our zero-aliasing correlation filter (ZACF) formulation, starting with the well-known MACE filter [9]. Because the MACE filter was one of the fundamental filters of the field, this problem exists in other formulations, including, but not limited to, the designs presented in [7, 10, 11, 12, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23]. Furthermore, any method using CFs, up until now, would likely benefit from incorporating our ZACF designs.

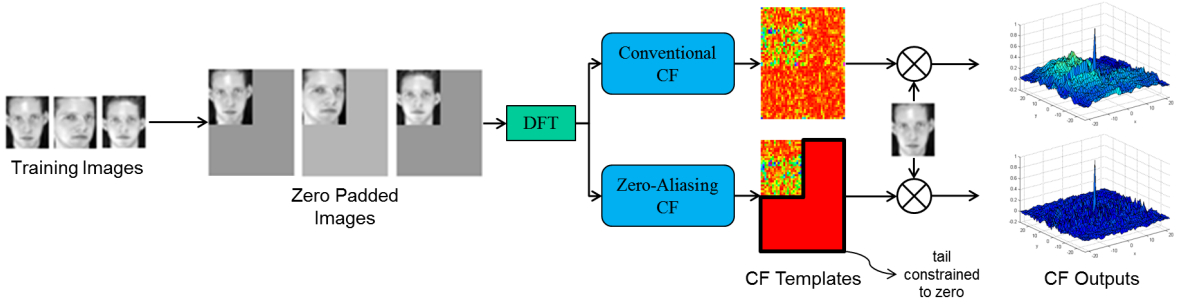


Figure 3.1: Overview of our proposed zero-aliasing approach. Conventional CF designs result in templates that are non-zero for all values. This means that, during the optimization, the correlation between the template and the test images is in fact a circular correlation. By constraining the tail of the template using zero-aliasing constraints, we force the optimization step to correspond to a linear correlation. This results in correlation planes that resemble the original design criteria - in this case, a sharp peak with low sidelobes.

3.1 Zero-Aliasing MACE Formulation

In this section, we present the formulation for the zero-aliasing MACE (ZAMACE) filter. The problem with the conventional FDMACE formulation is twofold. First, the CF template that is generated is the same size as the DFT used to train the CF. Typically, we desire a template that is the same size as the training signals (i.e., the template should be the same size as the signal/object of interest). While it is possible for the DFT size to be the same size as the training signals, this is not optimal, because the training images should be zero padded to avoid circular correlation. Therefore, it is clear that the training signals should be padded with zeros. The second problem with conventional CF designs is that there is no mechanism for forcing the tail of the CF template to zero (see Appendix B for further discussion). This is analogous to the zero-padding we enforce on the training signals. These two mechanisms work together to ensure that the correlation between the template and the training images is linear, i.e. there is no aliasing. A block diagram illustrates these ideas in Fig. 3.1. Note that our zero-aliasing formulation results in a template with a tail constrained to zero. As a result, the correlation output with a train image used as a test image will be much more desirable (i.e., a sharper peak with lower sidelobes). The details of how we achieve this follow.

The conventional FDMACE formulation solves for the CF $\bar{\mathbf{h}}$ in the frequency domain by minimizing the ACE term $\bar{\mathbf{h}}^{+}\mathbf{D}\bar{\mathbf{h}}$ subject to $\bar{\mathbf{X}}^{+}\bar{\mathbf{h}} = N\mathbf{u}$. However, this objective function (the expression of the ACE term) implicitly assumes circular correlation. To move towards a new formulation, we want a *time domain* template such that $h(n) = 0$ for $n \geq N_x$. Recall that the N-point IDFT is given by

$$h(n) = \frac{1}{N}\sum_{k=0}^{N-1} H(k)e^{j2\pi kn/N} \tag{3.1}$$

where $H(k)$ (unknown) denotes the kth elements of the CF in the frequency domain ($\bar{\mathbf{h}}$). We therefore want to satisfy the set of linear equations,

$$0 = \sum_{k=0}^{N-1} H(k)e^{j2\pi kn/N} \qquad , n \geq N_x \tag{3.2}$$

We can write this as

$$\mathbf{A}^{+}\bar{\mathbf{h}} = \mathbf{0} \tag{3.3}$$

where $\mathbf{0}$ is the $(N - N_x) \times 1$ zero vector and

$$\mathbf{A}^{+} = \begin{bmatrix} 1 & e^{j2\pi(1)(N_x)/N} & \cdots & e^{j2\pi(N-1)(N_x)/N} \\ 1 & e^{j2\pi(1)(N_x+1)/N} & \cdots & e^{j2\pi(N-1)(N_x+1)/N} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & e^{j2\pi(1)(N-1)/N} & \cdots & e^{j2\pi(N-1)(N-1)/N} \end{bmatrix} \tag{3.4}$$

Note that the matrix $\mathbf{A}^{+}$ is $(N - N_x) \times N$. We may now rewrite the optimization problem as the minimization of Eq. 2.6 subject to the old constraints (Eq. 2.8) and the new zero-aliasing constraints (Eq. 3.3). We can rewrite these constraints as

$$\mathbf{B}^{+}\bar{\mathbf{h}} = \mathbf{k} \tag{3.5}$$

where

$$\mathbf{B}^{+} = \begin{bmatrix} \bar{\mathbf{X}}^{+} \\ \mathbf{A}^{+} \end{bmatrix} \tag{3.6}$$

and

$$\mathbf{k} = \begin{bmatrix} N\mathbf{u} \\ \mathbf{0} \end{bmatrix} \tag{3.7}$$

Since this is in the same form as the FDMACE, the new ZAMACE formulation is given by

$$\bar{\mathbf{h}} = \mathbf{D}^{-1}\mathbf{B}\left(\mathbf{B}^{+}\mathbf{D}^{-1}\mathbf{B}\right)^{-1}\mathbf{k} \tag{3.8}$$

Note that $\mathbf{D}$ is $N \times N$, $\mathbf{B}$ is $N \times (Q + N - N_x)$, and $\mathbf{k}$ is $(Q + N - N_x) \times 1$. Note that the matrix $\mathbf{B}^{+}\mathbf{D}^{-1}\mathbf{B}$ is $(Q + N - N_x) \times (Q + N - N_x)$, and must be inverted. This is one downside of this proposed method, because the size of this matrix depends on the amount of zero padding $p = N - N_x$, and therefore grows with the signal's dimension. By comparison, the dimensionality of the matrix to be inverted in the standard FDMACE formulation was $Q \times Q$, which scales only by the number of training signals, making it more computationally feasible.

Let us return to the 1D example from the previous chapter to illustrate our zero-aliasing approach. Recall that we train CFs using $Q = 10$ heartbeat signals of length $N_x = 301$. Regardless of the training signal pad size (p), the conventional FDMACE formulation provides an unacceptable solution to this problem. This is because increasing p alone will never remove aliasing, as the template will always be nonzero for all $N_x + p$ values, and therefore the ACE term will

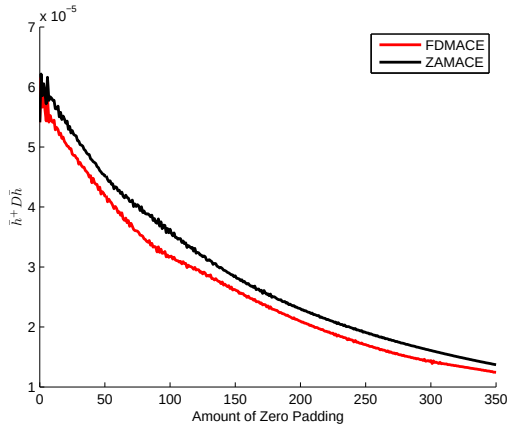
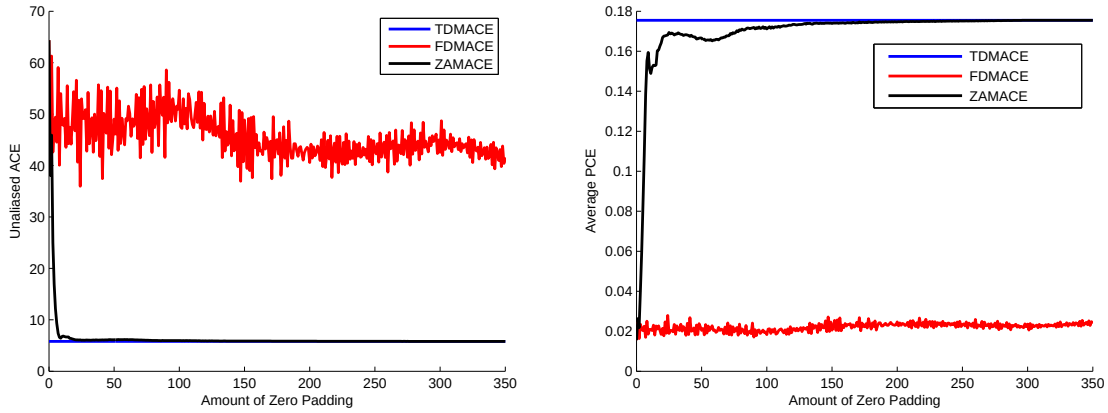


Figure 3.2: Value of $ACE = \bar{\mathbf{h}}^{+}\mathbf{D}\bar{\mathbf{h}}$ as a function of the amount of zero padding for both the FDMACE and ZAMACE formulations.

correspond to a circular correlation rather than a linear correlation. Using our example, We compare both the conventional and ZAMACE formulations while varying p. Because the ZAMACE formulation introduces additional constraints on $\bar{\mathbf{h}}$, we cannot expect the ACE term that we minimize ($\bar{\mathbf{h}}^{+}\mathbf{D}\bar{\mathbf{h}}$), to be any smaller. In Fig. 3.2 we show ACE as a function of zero padding. Notice that, as zero padding is increased, ACE decreases for both the FDMACE and ZAMACE formulations. However, ACE is always smaller for the original FDMACE formulation.

Despite this result, the $\bar{\mathbf{h}}^{+}\mathbf{D}\bar{\mathbf{h}}$ term (ACE) is not a good measure of unaliased ACE, which is what we really want to minimize. Specifically, the term $\bar{\mathbf{h}}^{+}\mathbf{D}\bar{\mathbf{h}}$ represents the average correlation energy of a *circular correlation* for the conventional FDMACE formulation. On the other hand, $\bar{\mathbf{h}}^{+}\mathbf{D}\bar{\mathbf{h}}$ represents the average correlation energy of a *linear correlation* for the ZAMACE formulation. This is the desired paradigm, as we apply the filter using linear correlations.

To illustrate the benefits of the zero-aliasing formulation, we compute FDMACE and ZAMACE templates and truncate them to length N_x. This is because we desire a template that is equal in size to the training signals. This cropping inconsequential for the ZAMACE because the last p values are zero anyways. For the FDMACE formulation, this cropping does not significantly affect performance (in fact it usually helps to crop the template). We then compare these templates to TDMACE, normalizing all templates to unit energy. We then compute the un-



(a) Unaliased ACE (b) PCE

Figure 3.3: Comparison of FDMACE, TDMACE, and ZAMACE. Note that the PCE values here appear to be small because we do not normalize by the number of elements in the correlation output. In this case, a PCE of ~0.18 corresponds to a very sharp peak.

aliased ACE (described in the previous experiments in the previous chapter) on the training set for each of the formulations. The results of this analysis are shown in Fig. 3.3a. Here note that the unaliased ACE for the conventional FDMACE formulation does not reduce as zero padding increases. The zero-aliasing approach, however, features an unaliased ACE that drops off considerably as zero padding increases. The unaliased ACE becomes the same as the TDMACE unaliased ACE at $p \geq 300$. We also compute the PCE (Eq. 2.1) for the correlation planes of each of the training images. We plot the average PCE for the training set in Fig. 3.3b. Note that the PCE for the zero-aliasing formulation increases as zero padding increases. However, note that PCE is quite good even for a small amount of zero-padding, which suggests that the ZAMACE filter will perform very well even when it does not completely eliminate all aliasing (we will take advantage of this fact later, in Section 4.1).

We also compute the resulting MSE between each of the ZAMACE and FDMACE templates and the TDMACE template in Fig. 3.4. Note that the ZAMACE template converges to the TDMACE template as the zero padding approaches 300. Note that the ZAMACE formulation acts as a bridge between the FDMACE and TDMACE formulations. When $p = 0$, the ZAMACE

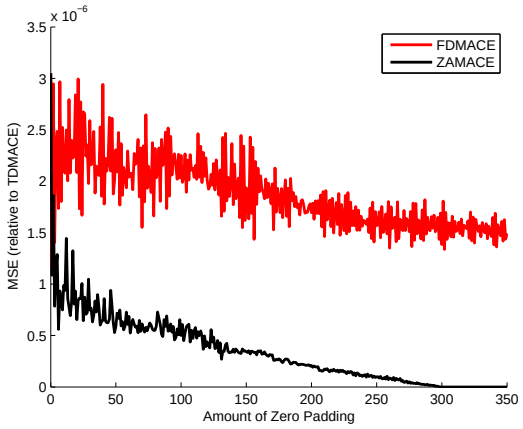


Figure 3.4: MSE of the FDMACE and ZAMACE filters compared to the TDMACE. Based on the magnitude of the training signals, an MSE on the order of 10^{-6} (in this example) is significant.

is equivalent to the FDMACE; when $p \geq 2N_x - 1$, the ZAMACE is equivalent to the TDMACE.

3.2 Extension to 2D

We can extend the formulation of the ZAMACE to 2D. In this case, the training images are of size $N_{x,1} \times N_{x,2}$, and the DFT size required to avoid aliasing is $N_1 \times N_2 = (2N_{x,1} - 1) \times (2N_{x,2} - 1)$. This is illustrated in Fig. 3.5. Here, the shaded region must be set to zero via zero-aliasing constraints.

In the 2D case, we must constrain the spatial values of the 2D IDFT of the frequency domain filter, $\bar{\mathbf{h}}$. The the $N_1 \times N_2$ 2D DFT is given by

$$h(n,m) = \frac{1}{N_1 N_2} \sum_{l=0}^{N_2-1} \left[\sum_{k=0}^{N_1-1} H(k,l) e^{j2\pi kn/N_1} \right] e^{j2\pi lm/N_2} \tag{3.9}$$

When we extend the FDMACE formulation to 2D, we vectorize each 2D DFT. For example, in Eq. 2.7, the matrix $\bar{\mathbf{X}}_q$ is the matrix formed by putting the vectorized 2D DFT of the training image $x_q(n,m)$ along the diagonal of an $N_1 \times N_2$ matrix. Similarly, the matrix $\bar{\mathbf{X}}$ in Eq. 2.8 is a matrix whose columns consist of the vectorized 2D DFTs of each of the training images. Finally, the correlation filter $\bar{\mathbf{h}}$ is the vectorized 2D DFT of the resulting $N_1 \times N_2$ correlation filter.

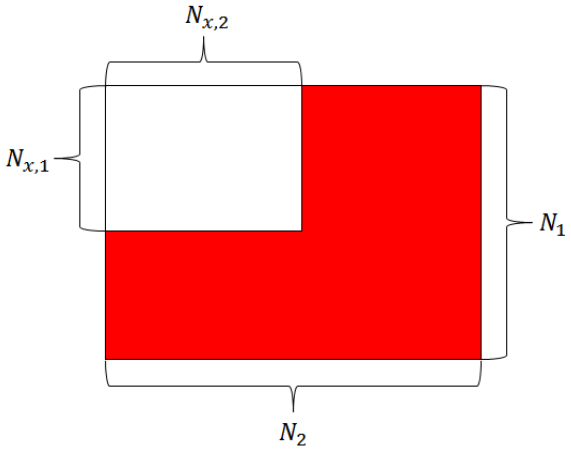


Figure 3.5: Illustration of the constraints required for the 2D formulation of the ZAMACE template. The shaded red region denotes the locations that are constrained.

Because our new formulation is based on how we vectorize each DFT, we must explicitly define a convention. Here, we will assume that we vectorize each DFT column-by-column. Written explicitly,

$$\bar{\mathbf{h}} = \begin{bmatrix} H(0,0) \\ \vdots \\ H(N_1-1,0) \\ H(0,1) \\ \vdots \\ H(N_1-1,1) \\ \vdots \\ H(0,N_2-1) \\ \vdots \\ H(N_1-1,N_2-1) \end{bmatrix} \tag{3.10}$$

We now seek a formulation that will translate the frequency domain elements of vector $\bar{\mathbf{h}}$ into spatial domain elements. Our first step is to compute the IDFT of each column,

$$\bar{\mathbf{h}}' = \mathbf{\Phi}\bar{\mathbf{h}} \tag{3.11}$$

where

$$\mathbf{\Phi} = \frac{1}{N_1}\begin{bmatrix} \mathbf{W}_{N_1}^* & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{W}_{N_1}^* & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{W}_{N_1}^* \end{bmatrix} \tag{3.12}$$

and $\mathbf{W}_{N_1}$ is the N_1 point DFT matrix. Next, we must compute the IDFT along each row. We define an intermediate matrix $\mathbf{\Psi}_m$, whose first row is given by

$$\mathbf{\Psi}_m(0,:) = \frac{1}{N_2}\begin{bmatrix} e^{j2\pi(0)m/N_2} & \mathbf{0}_{N_1-1}^T & e^{j2\pi(1)m/N_2} & \mathbf{0}_{N_1-1}^T & \cdots & e^{j2\pi(N_2-1)m/N_2} & \mathbf{0}_{N_1-1}^T \end{bmatrix} \tag{3.13}$$

where $\mathbf{0}_{N_1-1}$ is an $(N_1-1)\times 1$ zero vector. We can define the remaining rows of $\mathbf{\Psi}_m$ as

$$\mathbf{\Psi}_m(r,:) = CS(\mathbf{\Psi}_m(0,:), r) \qquad , r = 1, \ldots N_1 - 1 \tag{3.14}$$

where $CS(\mathbf{\Psi}_m(0,:), v)$ denotes a v-element right circular shift of the row vector $\mathbf{\Psi}_m(0,:)$. We then form the full matrix $\mathbf{\Omega}$ such that

$$\mathbf{\Omega} = \begin{bmatrix} \mathbf{\Psi}_0 \\ \mathbf{\Psi}_1 \\ \vdots \\ \mathbf{\Psi}_{N_2-1} \end{bmatrix} \tag{3.15}$$

Finally, we can compute the (spatial domain) vectorized 2D IDFT of $\bar{\mathbf{h}}$, as

$$\mathbf{h} = \mathbf{\Omega\Phi}\bar{\mathbf{h}} \tag{3.16}$$

To elaborate on this derivation, note that each $\mathbf{\Psi}_m$ effectively computes the mth element of the N_2-point IDFT for each row of the unvectorized 2D array $\bar{\mathbf{h}}'$. The reason for the zeros in the expression for each row of $\mathbf{\Psi}_m$ is because the row elements have been "split up" due to the vectorization. The reason for the circular shift is to "shift" the operators to work on each consecutive row of the unvectorized $\bar{\mathbf{h}}'$. It is often helpful to remind oneself of the structure of $\bar{\mathbf{h}}$ (Eq. 3.10) to understand these expressions. To illustrate, the second row of $\mathbf{\Psi}_m$ computes the first element of the IDFT for the second row of the unvectorized 2D array $\bar{\mathbf{h}}'$.

Note that Eq. 3.16 only gives the vectorized 2D IDFT of $\bar{\mathbf{h}}$. However, we only wish to constrain some of these values, specifically those illustrated in Fig. 3.5. Therefore, we must select the rows of the matrix $\mathbf{\Omega\Phi}$ corresponding to the values of $\mathbf{h}$ that we wish to constrain. We may do this by forming a matrix $\mathbf{S}$ such that

$$\mathbf{S} = \begin{bmatrix} \mathbf{0}_{N_{x,1}\times N_{x,2}} & \mathbf{1}_{N_{x,1}\times(N_2-N_{x,2})} \\ \mathbf{1}_{(N_1-N_{x,1})\times N_{x,2}} & \mathbf{1}_{(N_1-N_{x,1})\times(N_2-N_{x,2})} \end{bmatrix} \tag{3.17}$$

We may then define the selection vector $\mathbf{s}_v$ as

$$\mathbf{s}_v = vect(\mathbf{S}) \tag{3.18}$$

where $vect()$ is the column-by-column vectorization operation. We finally form a new matrix $\mathbf{A}^+$ whose rows are taken from the rows of matrix $\mathbf{\Omega\Phi}$ corresponding to entries in vector $\mathbf{s}_v$ that are equal to 1. At this point in time, the formulation of ZAMACE is identical to the 1D case, where we use Eq. 3.5 for the constraints and Eq. 3.8 to solve for the ZAMACE filter. However, notice in this case that matrix $\mathbf{D}$ is $N_1N_2 \times N_1N_2$, $\mathbf{B}$ is $N_1N_2 \times (Q + N_1N_2 - N_{x,1}N_{x,2})$, and $\mathbf{k}$ is $(Q + N_1N_2 - N_{x,1}N_{x,2}) \times 1$. Note that this means that the matrix to be inverted, $\mathbf{B}^+\mathbf{D}^{-1}\mathbf{B}$, is $(Q + N_1N_2 - N_{x,1}N_{x,2}) \times (Q + N_1N_2 - N_{x,1}N_{x,2})$. Therefore, as in the 1D case, the complexity

of this matrix inversion increases with dimensionality. However, note that the matrix

$$\mathbf{K} = \mathbf{B}^{+}\mathbf{D}^{-1}\mathbf{B} \tag{3.19}$$

is a real, symmetric, and positive definite matrix (see Appendix C). Because of this, we may decompose it using Cholesky factorization,

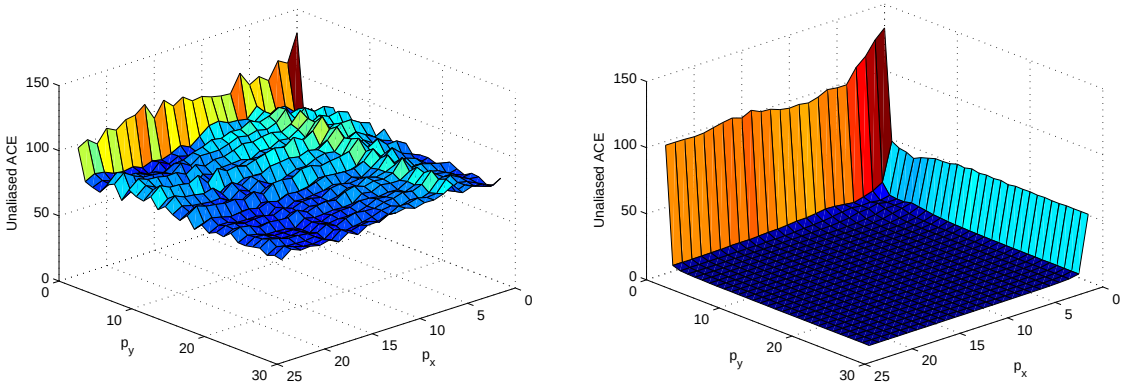
$$\mathbf{K} = \mathbf{R}^{T}\mathbf{R} \tag{3.20}$$

where $\mathbf{R}$ is an upper-triangular matrix of size $(Q + N^{\#} - N_x^{\#}) \times (Q + N^{\#} - N_x^{\#})$ in the general D-dimensional case. Therefore, the solution to ZAMACE can be written as

$$\begin{aligned} \bar{\mathbf{h}} &= \mathbf{D}^{-1}\mathbf{B}\left(\mathbf{R}^{T}\mathbf{R}\right)^{-1}\mathbf{k} \\ &= \mathbf{D}^{-1}\mathbf{B}\mathbf{R}^{-1}(\mathbf{R}^{T})^{-1}\mathbf{k} \end{aligned} \tag{3.21}$$

To solve for $\bar{\mathbf{h}}$, we may first compute $\mathbf{k}_1 = (\mathbf{R}^{T})^{-1}\mathbf{k}$ using forward substitution and then compute $\mathbf{k}_2 = \mathbf{R}^{-1}\mathbf{k}_1$ using back substitution. We can then compute $\bar{\mathbf{h}} = \mathbf{D}^{-1}\mathbf{B}\mathbf{k}_2$. This method is much faster and more memory efficient than computing the ZAMACE as in Eq. 3.8. In Chapter 4, we will discuss alternate formulations to the ZACF formulation that may be used to reduce to computational complexity.

To illustrate the ZAMACE in 2D, we consider a simple example. Here, we train both conventional and ZAMACE filters with three face images of the same person in the AT&T/ORL Database of Faces [43]. We have downsampled the images to size 28×23 pixels for computational reasons. In this case, we again vary the zero padding amount, except in this instance we vary it in both the vertical and horizontal dimensions, denoting it as p_1 and p_2, respectively. For each iteration, we use a DFT size of $N_1 \times N_2 = (N_{x,1} + p_1) \times (N_{x,2} + p_2)$. In each case, the resulting filter is cropped to size $N_{x,1} \times N_{x,2}$ and correlated with the training images using space



(a) Conventional MACE

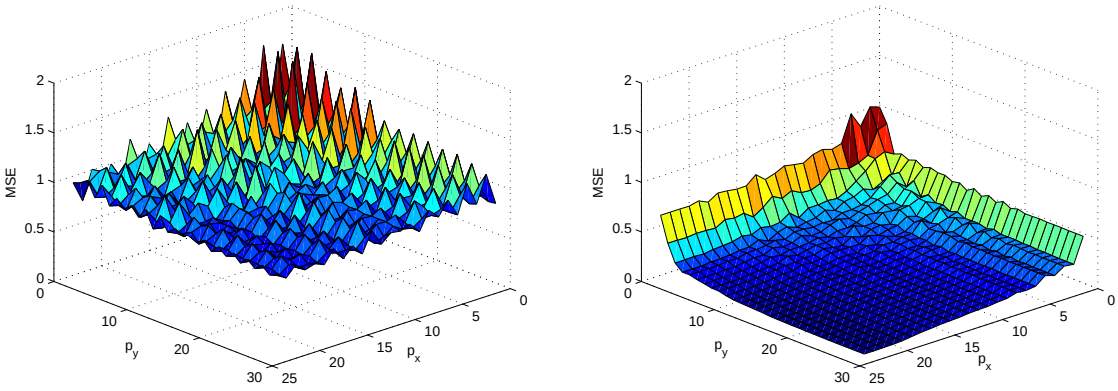
(b) ZAMACE

Figure 3.6: Plot of the unaliased ACE achieved on the training set for a 2D example, under different amounts of zero padding.

domain correlation. We then compute the unaliased ACE on the training set and show it in Fig. 3.6. Note that the conventional FDMACE formulation does not improve in terms of unaliased ACE as zero padding increases. The ZAMACE, on the other hand, features a sharp decrease in unaliased ACE as zero padding is increased. Like in the 1D case, note that just a small amount of zero padding leads to excellent results. This fact will be taken advantage of in Section 4.1. We also show corresponding plots for the MSE between the conventional FDMACE/ZAMACE and the space domain MACE template in Fig. 3.7. In the FDMACE case, the MSE decreases only slightly as zero padding increases, whereas MSE approaches zero for the ZAMACE case. Note that the amount of zero padding required to completely avoid aliasing is $p_1 = 27$ and $p_2 = 22$.

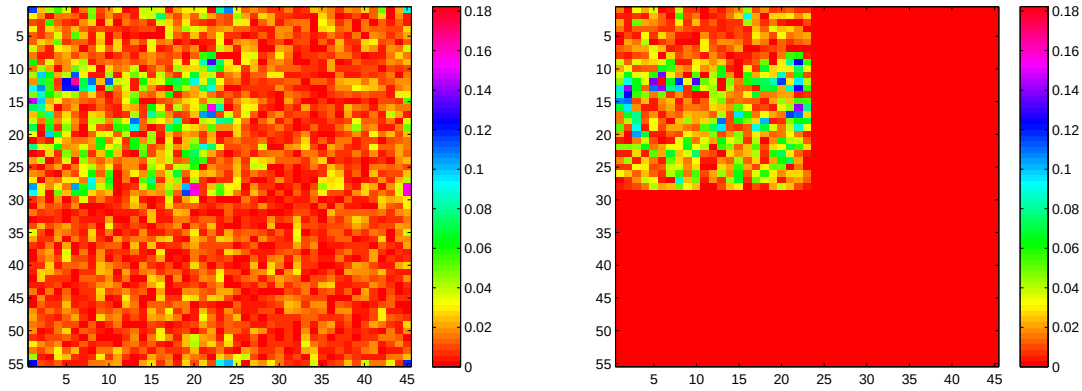
In Fig. 3.8, we show both the conventional and ZAMACE templates (uncropped, but normalized) at this level of zero padding. Note that the conventional FDMACE formulation results in some very high values outside of the image pixel ranges $y = (1, N_{x,1})$ and $x = (1, N_{x,2})$, whereas the ZAMACE does not.

Finally, we show an example correlation output obtained from correlating these templates with a training image in Fig. 3.9. Note that the correlation output has lower correlation energy for the zero-aliasing case than it does for the conventional case. The ZAMACE filter yields an



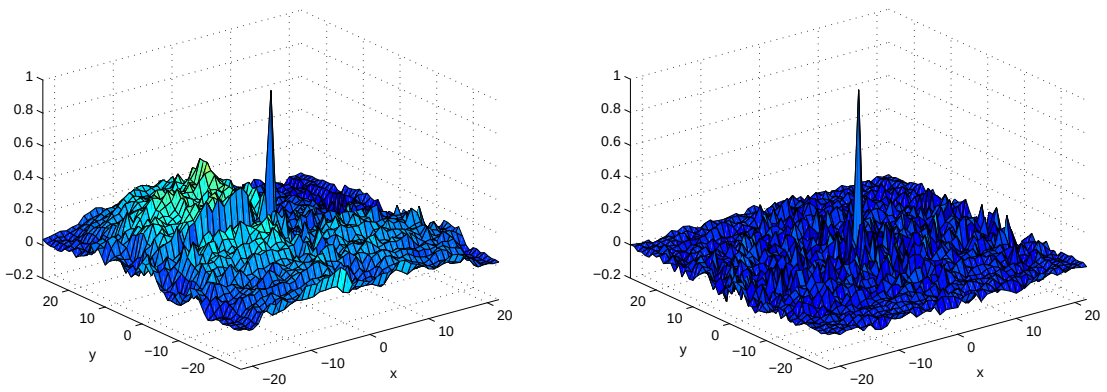
(a) Conventional MACE (b) ZAMACE

Figure 3.7: Plot of the MSE between each template and the space-domain MACE filter for a 2D example, under different amounts of zero padding. Based on the magnitude of the training images, an MSE on the order of 1 (in this example) is significant.



(a) Conventional MACE (b) ZAMACE

Figure 3.8: Space domain plots of the templates before truncation. Note that the conventional FDMACE has large values outside of the ranges of the desired template size, where as the ZA-MACE does not, as the constraints force these values to zero. In these plots, we display the absolute value of the templates.



(a) Conventional MACE

(b) ZAMACE

Figure 3.9: Example correlation outputs for one of the training images. Note that the ZAMACE filter yields an output that is much sharper and thus more consistent with the conventional FD-MACE filter design criteria.

output that is much more consistent with the original MACE filter design criteria.

We have demonstrated the fundamental ideas behind ZACFs by illustrating them explicitly using the TDMACE, FDMACE, and ZAMACE filters. We now extend these zero-aliasing constraints to other popular CF designs.

3.3 OTSDF

The OTSDF [14, 15] is a popular filter that builds on the ideas in the MACE filter. It combines the MACE filter with the MVSDF [13]. The MVSDF filter assumes that the input training images are corrupted by additive noise, and is designed such that the filter minimizes the ONV due to this noise. For a stationary noise process, the variance of the output noise is given by

$$\text{ONV} = \bar{\mathbf{h}}^{+}\mathbf{P}\bar{\mathbf{h}} \tag{3.22}$$

where the diagonal matrix $\mathbf{P}$ contains the power spectral density of the noise.

We may form a tradeoff between minimizing the ONV and ACE by minimizing a weighted

sum of the two criteria [14] as follows,

$$\begin{aligned}
&\min_{\bar{\mathbf{h}}} \lambda \text{ACE} + (1-\lambda)\text{ONV} \\
&\min_{\bar{\mathbf{h}}} \lambda \bar{\mathbf{h}}^{+}\mathbf{D}\bar{\mathbf{h}} + (1-\lambda)\bar{\mathbf{h}}^{+}\mathbf{P}\bar{\mathbf{h}} \\
&\min_{\bar{\mathbf{h}}} \bar{\mathbf{h}}^{+}\left(\lambda\mathbf{D} + (1-\lambda)\mathbf{P}\right)\bar{\mathbf{h}} \\
&\min_{\bar{\mathbf{h}}} \bar{\mathbf{h}}^{+}\mathbf{T}\bar{\mathbf{h}}
\end{aligned} \tag{3.23}$$

where

$$\mathbf{T} = \lambda\mathbf{D} + (1-\lambda)\mathbf{P} \tag{3.24}$$

At the same time, we include the peak constraints (Eq. 2.8) as in the MACE formulation. This yields the OTSDF filter,

$$\bar{\mathbf{h}} = \mathbf{T}^{-1}\bar{\mathbf{X}}\left(\bar{\mathbf{X}}^{+}\mathbf{T}^{-1}\bar{\mathbf{X}}\right)^{-1}\mathbf{u} \tag{3.25}$$

To derive the zero-aliasing OTSDF (ZAOTSDF), we simply include the zero-aliasing constraints (Eq. 3.3) in the minimization of Eq. 3.23. Similar to the ZAMACE filter, the ZAOTSDF filter is given by

$$\bar{\mathbf{h}} = \mathbf{T}^{-1}\mathbf{B}\left(\mathbf{B}^{+}\mathbf{T}^{-1}\mathbf{B}\right)^{-1}\mathbf{k} \tag{3.26}$$

Note that the matrix $\mathbf{B}^{+}\mathbf{T}^{-1}\mathbf{B}$ is positive definite, real, and symmetric, and $\bar{\mathbf{h}}$ may be computed efficiently in a similar manner to that described in Eq. 3.19 to Eq. 3.21.

3.4 Generalized Correlation Filters

Rodriguez and Kumar [7] proposed a unified framework to combine all constrained and unconstrained correlation filters into one framework. For constrained filters, the CCF framework is used, and for unconstrained filters, the UCF framework is used. Each of these frameworks minimize three metrics. By combining these parameters in different ways, the solution for the CCF/UCF filters simplifies to the solution for different constrained/unconstrained filter designs. Therefore, if we derive the zero-aliasing CCF (ZACCF) and the zero-aliasing UCF (ZAUCF) filters, we can essentially generalize zero-aliasing constraints to the majority of existing CF designs. Before we begin, we define the MSE, ASM, and ONV metrics. Then, we provide the derivations of UCF, ZAUCF, CCF, and ZACCF.

3.4.1 Preliminaries

First we define the MSE metric. The correlation output in the frequency domain in response to the qth training image is given by $\bar{\mathbf{g}}_q = \bar{\mathbf{X}}_q^+ \bar{\mathbf{h}}$. The *desired* correlation output in the frequency domain for the qth training image is $\bar{\mathbf{g}}_{D,q}$. The MSE metric is defined as

$$\text{MSE} = \frac{1}{NQ} \sum_{q=1}^{Q} \|\bar{\mathbf{g}}_q - \bar{\mathbf{g}}_{D,q}\|^2 \tag{3.27}$$

where N is the size of the DFT (we use 1D notation for simplicity). By minimizing the MSE metric, we are in effect attempting to make the actual correlation output look like the desired correlation output. We can expand the MSE expression in Eq. 3.27 as

$$\text{MSE} = \frac{1}{NQ}\sum_{q=1}^{Q} \bar{\mathbf{g}}_q^+ \bar{\mathbf{g}}_q - 2\bar{\mathbf{g}}_q^+ \bar{\mathbf{g}}_{D,q} + \bar{\mathbf{g}}_{D,q}^+ \bar{\mathbf{g}}_{D,q}$$

$$= \frac{1}{NQ}\sum_{q=1}^{Q} \bar{\mathbf{h}}^+ \bar{\mathbf{X}}_q \bar{\mathbf{X}}_q^* \bar{\mathbf{h}} - 2\bar{\mathbf{h}}^+ \bar{\mathbf{X}}_q \bar{\mathbf{g}}_{D,q} + \bar{\mathbf{g}}_{D,q}^+ \bar{\mathbf{g}}_{D,q}$$

$$= \bar{\mathbf{h}}^+ \left(\frac{1}{NQ}\sum_{q=1}^{Q} \bar{\mathbf{X}}_q \bar{\mathbf{X}}_q^* \right) \bar{\mathbf{h}} - 2\bar{\mathbf{h}}^+ \left(\frac{1}{NQ}\sum_{q=1}^{Q} \bar{\mathbf{X}}_q \bar{\mathbf{g}}_{D,q} \right) + \frac{1}{NQ}\sum_{q=1}^{Q} \bar{\mathbf{g}}_{D,q}^+ \bar{\mathbf{g}}_{D,q}$$

$$= \bar{\mathbf{h}}^+ \mathbf{D} \bar{\mathbf{h}} - 2\bar{\mathbf{h}}^+ \bar{\mathbf{p}} + E_f \tag{3.28}$$

where

$$\mathbf{D} = \frac{1}{NQ}\sum_{q=1}^{Q} \bar{\mathbf{X}}_q \bar{\mathbf{X}}_q^* \tag{3.29}$$

$$\bar{\mathbf{p}} = \frac{1}{NQ}\sum_{q=1}^{Q} \bar{\mathbf{X}}_q \bar{\mathbf{g}}_{D,q} \tag{3.30}$$

$$E_f = \frac{1}{NQ}\sum_{q=1}^{Q} \bar{\mathbf{g}}_{D,q}^+ \bar{\mathbf{g}}_{D,q} \tag{3.31}$$

Note that the MSE term contains the term $\bar{\mathbf{h}}^+ \mathbf{D} \bar{\mathbf{h}}$, which is the ACE as defined in the MACE formulation. We have repeated the expression here for convenience.

We also wish to design a filter that generalizes well to all images from a particular class. This approach was first presented in [16]. First, average squared error (ASE) is defined as

$$\text{ASE} = \frac{1}{Q}\sum_{q=1}^{Q} (\bar{\mathbf{g}}_q - \bar{\mathbf{f}})^+ (\bar{\mathbf{g}}_q - \bar{\mathbf{f}}) \tag{3.32}$$

where $\bar{\mathbf{g}}_q$ is the correlation output for training image q and $\bar{\mathbf{f}}$ is an (unknown) reference shape. We would like to choose $\bar{\mathbf{f}}$ to minimize ASE. This in effect makes all of the correlation outputs for one class similar by minimizing the variation between each correlation plane. We now take

the gradient of ASE in Eq. 3.32 with respect to $\bar{\mathbf{f}}$ and set to zero,

$$\begin{aligned}
\frac{2}{Q}\sum_{q=1}^{Q}(\bar{\mathbf{g}}_q - \bar{\mathbf{f}}) &= \mathbf{0} \\
\bar{\mathbf{f}} &= \frac{1}{Q}\sum_{q=1}^{Q}\bar{\mathbf{g}}_q \\
\bar{\mathbf{f}} &= \bar{\mathbf{g}}_{avg}
\end{aligned} \tag{3.33}$$

where $\bar{\mathbf{g}}_{avg} = \frac{1}{Q}\sum_{q=1}^{Q}\bar{\mathbf{g}}_q$ is the average correlation output plane. Note that $\bar{\mathbf{g}}_{avg}$ may be written as

$$\begin{aligned}
\bar{\mathbf{g}}_{avg} &= \frac{1}{Q}\sum_{q=1}^{Q}\bar{\mathbf{X}}_q^{+}\bar{\mathbf{h}} \\
&= \bar{\mathbf{X}}_{avg}^{+}\bar{\mathbf{h}}
\end{aligned} \tag{3.34}$$

where $\bar{\mathbf{X}}_{avg} = \frac{1}{Q}\sum_{q=1}^{Q}\bar{\mathbf{X}}_q$ is the average of the DFTs of the training images. We can now substitute the optimal $\bar{\mathbf{f}} = \bar{\mathbf{g}}_{avg}$ into the expression for ASE (Eq. 3.32). This yields what is defined as ASM (discussed in the previous chapter),

$$\text{ASM} = \frac{1}{NQ}\sum_{q=1}^{Q}(\bar{\mathbf{g}}_q - \bar{\mathbf{g}}_{avg})^{+}(\bar{\mathbf{g}}_q - \bar{\mathbf{g}}_{avg}) \tag{3.35}$$

The ASM metric represents a measure of how dissimilar correlation output $\bar{\mathbf{g}}_q$ is from the average correlation output $\bar{\mathbf{g}}_{avg}$. By minimizing ASM, we are in effect minimizing the differences between the correlation outputs $\bar{\mathbf{g}}_q$ for $q = 1 \ldots Q$. The CF can also be interpreted as a transform that maximizes the compactness of each class in the transform domain (in this case, the transform domain is the correlation domain). Expanding Eq. 3.36 gives

$$\text{ASM} = \frac{1}{NQ}\sum_{q=1}^{Q}\left(\bar{\mathbf{g}}_q^+\bar{\mathbf{g}}_q - \bar{\mathbf{g}}_q^+\bar{\mathbf{g}}_{avg} - \bar{\mathbf{g}}_{avg}^+\bar{\mathbf{g}}_q + \bar{\mathbf{g}}_{avg}^+\bar{\mathbf{g}}_{avg}\right)$$

$$= \frac{1}{NQ}\sum_{q=1}^{Q}\bar{\mathbf{g}}_q^+\bar{\mathbf{g}}_q - \frac{1}{N}\left(\frac{1}{Q}\sum_{q=1}^{Q}\bar{\mathbf{g}}_q^+\right)\bar{\mathbf{g}}_{avg} - \frac{1}{N}\bar{\mathbf{g}}_{avg}^+\left(\frac{1}{Q}\sum_{q=1}^{Q}\bar{\mathbf{g}}_q\right) + \frac{1}{NQ}\sum_{q=1}^{Q}\bar{\mathbf{g}}_{avg}^+\bar{\mathbf{g}}_{avg}$$

$$= \frac{1}{NQ}\sum_{q=1}^{Q}\bar{\mathbf{g}}_q^+\bar{\mathbf{g}}_q - \frac{1}{N}\bar{\mathbf{g}}_{avg}^+\bar{\mathbf{g}}_{avg} - \frac{1}{N}\bar{\mathbf{g}}_{avg}^+\bar{\mathbf{g}}_{avg} + \frac{1}{N}\bar{\mathbf{g}}_{avg}^+\bar{\mathbf{g}}_{avg}$$

$$= \frac{1}{NQ}\sum_{q=1}^{Q}\bar{\mathbf{g}}_q^+\bar{\mathbf{g}}_q - \frac{1}{N}\bar{\mathbf{g}}_{avg}^+\bar{\mathbf{g}}_{avg}$$

$$= \frac{1}{NQ}\sum_{q=1}^{Q}\bar{\mathbf{h}}^+\bar{\mathbf{X}}_q\bar{\mathbf{X}}_q^*\bar{\mathbf{h}} - \frac{1}{N}\bar{\mathbf{h}}^+\bar{\mathbf{X}}_{avg}\bar{\mathbf{X}}_{avg}^+\bar{\mathbf{h}}$$

$$= \bar{\mathbf{h}}^+\left(\frac{1}{NQ}\sum_{q=1}^{Q}\bar{\mathbf{X}}_q\bar{\mathbf{X}}_q^*\right)\bar{\mathbf{h}} - \bar{\mathbf{h}}^+\left(\frac{1}{N}\bar{\mathbf{X}}_{avg}\bar{\mathbf{X}}_{avg}^+\right)\bar{\mathbf{h}}$$

$$= \bar{\mathbf{h}}^+\mathbf{D}\bar{\mathbf{h}} - \bar{\mathbf{h}}^+\mathbf{M}\bar{\mathbf{h}} \tag{3.36}$$

where

$$\mathbf{M} = \frac{1}{N}\bar{\mathbf{X}}_{avg}\bar{\mathbf{X}}_{avg}^+ \tag{3.37}$$

Finally, we may write the ONV as previously defined for the OTSDF filter. We repeat it here for convenience,

$$\text{ONV} = \bar{\mathbf{h}}^+\mathbf{P}\bar{\mathbf{h}} \tag{3.38}$$

where $\mathbf{P}$ is the power spectral density of the input noise. Note that $\mathbf{D}$, $\mathbf{M}$, and $\mathbf{P}$ are all diagonal matrices. We now present the UCF and CCF formulations. These formulations will minimize a combination of the quadratic terms MSE, ASM, and ONV. This is similar to OTSDF, in which the weighted sum of two quadratics was minimized. The primary difference between these for-

mulations is that the UCF will have no peak constraints on the optimization, whereas the CCF will enforce peak constraints.

3.4.2 UCF

The UCF filter [7] is derived by minimizing the functional

$$\begin{aligned}\mathcal{L}_{UCF}(\bar{\mathbf{h}},\gamma,\beta) &= \text{MSE} + \gamma\text{ASM} + \beta\text{ONV} \\ &= \bar{\mathbf{h}}^{+}\mathbf{D}\bar{\mathbf{h}} - 2\bar{\mathbf{h}}^{+}\bar{\mathbf{p}} + E_f + \gamma\left(\bar{\mathbf{h}}^{+}\mathbf{D}\bar{\mathbf{h}} - \bar{\mathbf{h}}^{+}\mathbf{M}\bar{\mathbf{h}}\right) + \beta\bar{\mathbf{h}}^{+}\mathbf{P}\bar{\mathbf{h}} \\ &= \bar{\mathbf{h}}^{+}\left((1+\gamma)\mathbf{D} - \gamma\mathbf{M} + \beta\mathbf{P}\right)\bar{\mathbf{h}} - 2\bar{\mathbf{h}}^{+}\bar{\mathbf{p}} + E_f \\ &= \bar{\mathbf{h}}^{+}\mathbf{T}\bar{\mathbf{h}} - 2\bar{\mathbf{h}}^{+}\bar{\mathbf{p}} + E_f \end{aligned} \tag{3.39}$$

where

$$\mathbf{T} = (1+\gamma)\mathbf{D} - \gamma\mathbf{M} + \beta\mathbf{P} \tag{3.40}$$

and γ and β are Lagrange multipliers. We want to minimize $\mathcal{L}_{UCF}(\bar{\mathbf{h}},\gamma,\beta)$, so we take the gradient with respect to $\bar{\mathbf{h}}$ and set it to zero,

$$\frac{\partial\mathcal{L}_{UCF}}{\partial\bar{\mathbf{h}}} = 2\mathbf{T}\bar{\mathbf{h}} - 2\bar{\mathbf{p}} = \mathbf{0} \tag{3.41}$$

Finally we can solve for the UCF filter,

$$\bar{\mathbf{h}} = \mathbf{T}^{-1}\bar{\mathbf{p}} \tag{3.42}$$

3.4.3 ZAUCF

To derive the ZAUCF, we minimize $\mathcal{L}_{UCF}(\bar{\mathbf{h}}, \gamma, \beta)$ subject to the zero-aliasing constraints, $\mathbf{A}^{+}\bar{\mathbf{h}} = \mathbf{0}$. We do this by forming the new functional

$$\mathcal{L}_{ZAUCF}(\bar{\mathbf{h}}, \gamma, \beta, \boldsymbol{\omega}) = \bar{\mathbf{h}}^{+}\mathbf{T}\bar{\mathbf{h}} - 2\bar{\mathbf{h}}^{+}\bar{\mathbf{p}} + E_f - 2\boldsymbol{\omega}^{+}(\mathbf{A}^{+}\bar{\mathbf{h}})$$

where we have introduced the vector $\boldsymbol{\omega}$, which is a vector of Lagrangian multipliers. We want to minimize $\mathcal{L}_{ZAUCF}(\bar{\mathbf{h}}, \gamma, \beta, \boldsymbol{\omega})$, so we take the gradient with respect to $\bar{\mathbf{h}}$ and set it to zero,

$$\frac{\partial \mathcal{L}_{ZAUCF}}{\partial \bar{\mathbf{h}}} = 2\mathbf{T}\bar{\mathbf{h}} - 2\bar{\mathbf{p}} - 2\mathbf{A}\boldsymbol{\omega} = \mathbf{0} \tag{3.43}$$

We can solve for $\bar{\mathbf{h}}$,

$$\bar{\mathbf{h}} = \mathbf{T}^{-1}(\bar{\mathbf{p}} + \mathbf{A}\boldsymbol{\omega}) \tag{3.44}$$

We then substitute this back into the zero-aliasing constraint equation, and solve for $\boldsymbol{\omega}$,

$$\begin{aligned} \mathbf{A}^{+}(\mathbf{T}^{-1}(\bar{\mathbf{p}} + \mathbf{A}\boldsymbol{\omega})) &= \mathbf{0} \\ \mathbf{A}^{+}\mathbf{T}^{-1}\bar{\mathbf{p}} + \mathbf{A}^{+}\mathbf{T}^{-1}\mathbf{A}\boldsymbol{\omega} &= \mathbf{0} \\ \boldsymbol{\omega} &= -(\mathbf{A}^{+}\mathbf{T}^{-1}\mathbf{A})^{-1}\mathbf{A}^{+}\mathbf{T}^{-1}\bar{\mathbf{p}} \end{aligned} \tag{3.45}$$

Substituting this back into Eq. 3.44, we obtain the ZAUCF filter,

$$\begin{aligned}\bar{\mathbf{h}} &= \mathbf{T}^{-1}\bar{\mathbf{p}} - \mathbf{T}^{-1}\mathbf{A}(\mathbf{A}^{+}\mathbf{T}^{-1}\mathbf{A})^{-1}\mathbf{A}^{+}\mathbf{T}^{-1}\bar{\mathbf{p}} \\ &= \boldsymbol{\Delta}_{\mathbf{T}}\mathbf{T}^{-1}\bar{\mathbf{p}} \end{aligned} \tag{3.46}$$

where

$$\boldsymbol{\Delta}_{\mathbf{T}} = \mathbf{I} - \mathbf{T}^{-1}\mathbf{A}(\mathbf{A}^{+}\mathbf{T}^{-1}\mathbf{A})^{-1}\mathbf{A}^{+} \tag{3.47}$$

Note that the matrix $\mathbf{A}^{+}\mathbf{T}^{-1}\mathbf{A}$ is positive definite, real, and symmetric, and $\bar{\mathbf{h}}$ may be computed efficiently in a similar manner to that described in Eq. 3.19 to Eq. 3.21.

3.4.4 CCF

In the case of the CCF [7], the filter is derived by minimizing the linear combination MSE + γASM + βONV from the UCF formulation subject to the peak constraints $\bar{\mathbf{X}}^{+}\bar{\mathbf{h}} = N\mathbf{u}$. Therefore we form the new functional

$$\mathcal{L}_{CCF}(\bar{\mathbf{h}}, \gamma, \beta, \boldsymbol{\omega}) = \bar{\mathbf{h}}^{+}\mathbf{T}\bar{\mathbf{h}} - 2\bar{\mathbf{h}}^{+}\bar{\mathbf{p}} + E_f - 2\boldsymbol{\omega}^{+}(\bar{\mathbf{X}}^{+}\bar{\mathbf{h}} - N\mathbf{u}) \tag{3.48}$$

We want to minimize $\mathcal{L}_{CCF}(\bar{\mathbf{h}}, \gamma, \beta, \boldsymbol{\omega})$, so we take the gradient with respect to $\bar{\mathbf{h}}$ and set it to zero,

$$\frac{\partial \mathcal{L}_{CCF}}{\partial \bar{\mathbf{h}}} = 2\mathbf{T}\bar{\mathbf{h}} - 2\bar{\mathbf{p}} - 2\bar{\mathbf{X}}\boldsymbol{\omega} = \mathbf{0} \tag{3.49}$$

We can solve for $\bar{\mathbf{h}}$,

$$\bar{\mathbf{h}} = \mathbf{T}^{-1}(\bar{\mathbf{p}} + \bar{\mathbf{X}}\boldsymbol{\omega}) \tag{3.50}$$

we then substitute this back into the peak constraints equation, and solve for $\boldsymbol{\omega}$,

$$\begin{aligned}
\bar{\mathbf{X}}^{+}(\mathbf{T}^{-1}(\bar{\mathbf{p}} + \bar{\mathbf{X}}\boldsymbol{\omega})) &= N\mathbf{u} \\
\bar{\mathbf{X}}^{+}\mathbf{T}^{-1}\bar{\mathbf{p}} + \bar{\mathbf{X}}^{+}\mathbf{T}^{-1}\bar{\mathbf{X}}\boldsymbol{\omega} &= N\mathbf{u} \\
\boldsymbol{\omega} &= (\bar{\mathbf{X}}^{+}\mathbf{T}^{-1}\bar{\mathbf{X}})^{-1}(N\mathbf{u} - \bar{\mathbf{X}}^{+}\mathbf{T}^{-1}\bar{\mathbf{p}})
\end{aligned} \tag{3.51}$$

substituting this back into Eq. 3.50, we obtain the CCF filter,

$$\bar{\mathbf{h}} = \mathbf{T}^{-1}\bar{\mathbf{p}} + \mathbf{T}^{-1}\bar{\mathbf{X}}(\bar{\mathbf{X}}^{+}\mathbf{T}^{-1}\bar{\mathbf{X}})^{-1}(N\mathbf{u} - \bar{\mathbf{X}}^{+}\mathbf{T}^{-1}\bar{\mathbf{p}}) \tag{3.52}$$

3.4.5 ZACCF

To derive the ZACCF, we impose the additional zero-aliasing constraints. As in the ZAMACE case, we group the peak constraints $\bar{\mathbf{X}}^{+}\bar{\mathbf{h}} = N\mathbf{u}$ and the zero-aliasing constraints, $\mathbf{A}^{+}\bar{\mathbf{h}} = \mathbf{0}$ into a single constraint equation, $\mathbf{B}^{+}\bar{\mathbf{h}} = \mathbf{k}$. At this point, the ZACCF derivation is the same as that of the CCF, except that $\bar{\mathbf{X}}$ is replaced by $\mathbf{B}$ and $N\mathbf{u}$ is replaced by $\mathbf{k}$. The ZACCF is given by

$$\bar{\mathbf{h}} = \mathbf{T}^{-1}\bar{\mathbf{p}} + \mathbf{T}^{-1}\mathbf{B}(\mathbf{B}^{+}\mathbf{T}^{-1}\mathbf{B})^{-1}(\mathbf{k} - \mathbf{B}^{+}\mathbf{T}^{-1}\bar{\mathbf{p}}) \tag{3.53}$$

Note that the matrix $\mathbf{B}^{+}\mathbf{T}^{-1}\mathbf{B}$ is positive definite, real, and symmetric, and $\bar{\mathbf{h}}$ may be computed efficiently in a similar manner to that described in Eq. 3.19 to Eq. 3.21.

3.5 MACH

We now turn our attention to the MACH filter [16]. Previously, we stated that the UCF formulation can be used as a general case formulation for unconstrained CF designs. However, it is unclear if applying the zero-aliasing constraints to the UCF formulation will give the same result as applying the zero-aliasing constraints to the MACH filter formulation, because these formulations minimize different cost functions. In this section, we introduce the MACH filter formulation and then derive the zero-aliasing MACH (ZAMACH) filter. By doing this, we prove that the ZAMACH filter is of the same form as the ZAUCF.

3.5.1 MACH

The MACH filter is aimed at generalizing well to all images from a particular class. Instead of specifying the response to individual training images, as in constrained correlation filters, the MACH filter attempts to generalize over all the training images provided to it. The MACH filter comes in different forms, but here we will present the version that minimizes ACE and ASM. First, ASM (initially presented in Eq. 3.35) can be expanded as

$$\begin{aligned}
\text{ASM} &= \frac{1}{Q}\sum_{q=1}^{Q}(\bar{\mathbf{g}}_q - \bar{\mathbf{g}}_{avg})^{+}(\bar{\mathbf{g}}_q - \bar{\mathbf{g}}_{avg}) \\
&= \frac{1}{Q}\sum_{q=1}^{Q}(\bar{\mathbf{X}}_q^{+}\bar{\mathbf{h}} - \bar{\mathbf{X}}_{avg}^{+}\bar{\mathbf{h}})^{+}(\bar{\mathbf{X}}_q^{+}\bar{\mathbf{h}} - \bar{\mathbf{X}}_{avg}^{+}\bar{\mathbf{h}}) \\
&= \bar{\mathbf{h}}^{+}\left[\frac{1}{Q}\sum_{q=1}^{Q}(\bar{\mathbf{X}}_q - \bar{\mathbf{X}}_{avg})(\bar{\mathbf{X}}_q - \bar{\mathbf{X}}_{avg})^{+}\right]\bar{\mathbf{h}} \\
&= \bar{\mathbf{h}}^{+}\mathbf{S}\bar{\mathbf{h}} \qquad (3.54)
\end{aligned}$$

where

$$\mathbf{S} = \frac{1}{Q}\sum_{q=1}^{Q}(\bar{\mathbf{X}}_q - \bar{\mathbf{X}}_{avg})(\bar{\mathbf{X}}_q - \bar{\mathbf{X}}_{avg})^{+} \tag{3.55}$$

is a diagonal matrix that is a frequency domain measure of the similarity of the training images to the class mean. From before, note that $\mathbf{S} = \mathbf{D} - \mathbf{M}$. The MACH filter also minimizes ACE, which was introduced in Eq. 2.6. Because there are no peak constraints, the MACH filter makes the average correlation peak as large as possible. This is done by *maximizing* the average correlation height (ACH)[1], which is defined as

$$ACH = \left| \frac{1}{NQ} \sum_{q=1}^{Q} g_q(0,0) \right|^2 \tag{3.56}$$

where $g_q(0,0)$ is the spatial domain correlation value at the origin for the qth training image. In the frequency domain, this may be expressed as

$$\begin{aligned} ACH &= \left| \frac{1}{Q} \sum_{q=1}^{Q} \bar{\mathbf{x}}_q^{+} \bar{\mathbf{h}} \right|^2 \\ &= \left| \bar{\mathbf{m}}^{+} \bar{\mathbf{h}} \right|^2 \\ &= \bar{\mathbf{h}}^{+} \bar{\mathbf{m}} \bar{\mathbf{m}}^{+} \bar{\mathbf{h}} \end{aligned} \tag{3.57}$$

where $\bar{\mathbf{m}} = \frac{1}{Q}\sum_{q=1}^{Q} \bar{\mathbf{x}}_q$ is the mean of the vectorized 2D DFTs of the Q training images $\bar{\mathbf{x}}_q$, $q = 1 \ldots Q$. The MACH filter is formulated by maximizing ACH and minimizing ASM and ACE. This is accomplished by maximizing the Rayleigh quotient given by

[1]Strictly speaking, this is the square of the average correlation height. The form used for ACH here is used for dimensional consistency with ACE and ASM, which are quadratic functions of $\bar{\mathbf{h}}$.

$$
\begin{aligned}
J(\bar{\mathbf{h}}) &= \frac{\bar{\mathbf{h}}^{+}\bar{\mathbf{m}}\bar{\mathbf{m}}^{+}\bar{\mathbf{h}}}{\gamma\bar{\mathbf{h}}^{+}\mathbf{S}\bar{\mathbf{h}} + \alpha\bar{\mathbf{h}}^{+}\mathbf{D}\bar{\mathbf{h}}} \\
&= \frac{\bar{\mathbf{h}}^{+}\bar{\mathbf{m}}\bar{\mathbf{m}}^{+}\bar{\mathbf{h}}}{\bar{\mathbf{h}}^{+}(\gamma\mathbf{S} + \alpha\mathbf{D})\bar{\mathbf{h}}} \\
&= \frac{\bar{\mathbf{h}}^{+}\bar{\mathbf{m}}\bar{\mathbf{m}}^{+}\bar{\mathbf{h}}}{\bar{\mathbf{h}}^{+}\mathbf{T}\bar{\mathbf{h}}}
\end{aligned}
\tag{3.58}
$$

where

$$
\mathbf{T} = \gamma\mathbf{S} + \alpha\mathbf{D} \tag{3.59}
$$

Note that if additional metrics are minimized (such as ONV), they would simply be included in the expression for $\mathbf{T}$. To solve for the MACH filter, we take the gradient of Eq. 3.58 with respect to $\bar{\mathbf{h}}$ and set it to zero,

$$
\begin{aligned}
\frac{\bar{\mathbf{m}}\bar{\mathbf{m}}^{+}\bar{\mathbf{h}}(\bar{\mathbf{h}}^{+}\mathbf{T}\bar{\mathbf{h}}) - \mathbf{T}\bar{\mathbf{h}}(\bar{\mathbf{h}}^{+}\bar{\mathbf{m}}\bar{\mathbf{m}}^{+}\bar{\mathbf{h}})}{(\bar{\mathbf{h}}^{+}\mathbf{T}\bar{\mathbf{h}})^{2}} &= \mathbf{0} \\
\frac{1}{\bar{\mathbf{h}}^{+}\mathbf{T}\bar{\mathbf{h}}}\left(\bar{\mathbf{m}}\bar{\mathbf{m}}^{+}\bar{\mathbf{h}} - \frac{\bar{\mathbf{h}}^{+}\bar{\mathbf{m}}\bar{\mathbf{m}}^{+}\bar{\mathbf{h}}}{\bar{\mathbf{h}}^{+}\mathbf{T}\bar{\mathbf{h}}}\mathbf{T}\bar{\mathbf{h}}\right) &= \mathbf{0} \\
\bar{\mathbf{m}}\bar{\mathbf{m}}^{+}\bar{\mathbf{h}} &= J(\bar{\mathbf{h}})\mathbf{T}\bar{\mathbf{h}} \\
\mathbf{T}^{-1}\bar{\mathbf{m}}\bar{\mathbf{m}}^{+}\bar{\mathbf{h}} &= J(\bar{\mathbf{h}})\bar{\mathbf{h}}
\end{aligned}
\tag{3.60}
$$

Note that this is an eigenvalue problem. Here, $J(\bar{\mathbf{h}})$ acts as the eigenvalue in this expression. Therefore, the solution for the MACH filter is to choose the eigenvector of the matrix $\mathbf{T}^{-1}\bar{\mathbf{m}}\bar{\mathbf{m}}^{+}$ corresponding to the largest eigenvalue. Because $\bar{\mathbf{m}}\bar{\mathbf{m}}^{+}$ is of rank 1, there is only one eigenvalue. We may write Eq. 3.60 as

$$\mathbf{T}^{-1}\bar{\mathbf{m}}\lambda = J(\bar{\mathbf{h}})\bar{\mathbf{h}}$$

$$\mathbf{T}^{-1}\bar{\mathbf{m}} = \phi\bar{\mathbf{h}}$$

where $\lambda = \bar{\mathbf{m}}^{+}\bar{\mathbf{h}}$ is a constant and $\phi = J(\bar{\mathbf{h}})/\lambda$ is a constant scaling factor. We can ignore ϕ (because $J(\bar{\mathbf{h}})$ is unaffected by scalar multipliers) and write the MACH filter solution as

$$\bar{\mathbf{h}} = \mathbf{T}^{-1}\bar{\mathbf{m}} \tag{3.61}$$

Note that this solution is given by the UCF formulation (Eq. 3.42) when the desired correlation output is a delta function in the time/spatial domain. This means that $\bar{\mathbf{g}}_{D,q}$ will be a constant in the frequency domain, and $\bar{\mathbf{p}}$ reduces to the mean of the DFTs of the training images, $\bar{\mathbf{m}}$ (compare Eq. 3.42 to Eq. 3.61).

3.5.2 ZAMACH

In Section 3.4.2, we presented the UCF formulation. In Section 3.5, we showed that the MACH filter solution can be obtained from this UCF formulation. However, it is unclear if the ZAUCF (Section 3.4.3) gives the same solution as the the ZAMACH filter. This is because the ZAUCF formulation minimizes metrics (such as ACE and ASM) subject to zero-aliasing constraints. However, note that the MACH filter minimizes these same metrics *while also maximizing the ACH*.

To derive the ZAMACH filter, we maximize the original MACH cost function given in Eq. 3.58 subject to the zero-aliasing constraints $\mathbf{A}^{+}\bar{\mathbf{h}} = \mathbf{0}$. For convenience, we repeat Eq. 3.58 below,

$$J(\bar{\mathbf{h}}) = \frac{\bar{\mathbf{h}}^{+}\bar{\mathbf{m}}\bar{\mathbf{m}}^{+}\bar{\mathbf{h}}}{\bar{\mathbf{h}}^{+}\mathbf{T}\bar{\mathbf{h}}} \tag{3.62}$$

Because $\mathbf{T}$ is a real, positive diagonal matrix, we may express it as $\mathbf{T} = \mathbf{T}^{1/2}\mathbf{T}^{1/2}$. Through a change in variable $\mathbf{y} = \mathbf{T}^{1/2}\bar{\mathbf{h}}$, we can rewrite this as

$$\begin{aligned} J(\bar{\mathbf{h}}) &= \frac{\mathbf{y}^{+}\mathbf{T}^{-1/2}\bar{\mathbf{m}}\bar{\mathbf{m}}^{+}\mathbf{T}^{-1/2}\mathbf{y}}{\mathbf{y}^{+}\mathbf{y}} \\ &= \frac{\mathbf{y}^{+}\mathbf{\Lambda}\mathbf{y}}{\mathbf{y}^{+}\mathbf{y}} \end{aligned} \tag{3.63}$$

where

$$\mathbf{\Lambda} = \mathbf{T}^{-1/2}\bar{\mathbf{m}}\bar{\mathbf{m}}^{+}\mathbf{T}^{-1/2} \tag{3.64}$$

The zero-aliasing constraints $\mathbf{A}^{+}\bar{\mathbf{h}} = \mathbf{0}$ can also be rewritten as

$$\mathbf{C}^{+}\mathbf{y} = \mathbf{0} \tag{3.65}$$

where

$$\mathbf{C} = \mathbf{T}^{-1/2}\mathbf{A} \tag{3.66}$$

Before proceeding, we must ensure that the matrix $\mathbf{\Lambda}$ is positive semi-definite (we want to ensure that the numerator, $\mathbf{y}^{+}\mathbf{\Lambda}\mathbf{y}$ is a positive number before we maximize $J(\bar{\mathbf{h}})$). To show this, note that we can write

$$\begin{aligned}\mathbf{y}^{+}\mathbf{T}^{-1/2}\bar{\mathbf{m}}\bar{\mathbf{m}}^{+}\mathbf{T}^{-1/2}\mathbf{y} &= \mathbf{k}^{+}\bar{\mathbf{m}}\bar{\mathbf{m}}^{+}\mathbf{k} \\ &= \mathbf{k}^{+}\bar{\mathbf{m}}\bar{\mathbf{m}}^{+}\mathbf{k} \\ &= (\mathbf{k}^{+}\bar{\mathbf{m}})(\mathbf{k}^{+}\bar{\mathbf{m}})^{+} \\ &= (\mathbf{k}^{+}\bar{\mathbf{m}})(\mathbf{k}^{+}\bar{\mathbf{m}})^{*} \\ &= |\mathbf{k}^{+}\bar{\mathbf{m}}|^{2} \end{aligned} \tag{3.67}$$

where

$$\mathbf{k} = \mathbf{T}^{-1/2}\mathbf{y} \tag{3.68}$$

Note that $\mathbf{k}^{+}\bar{\mathbf{m}}$ is a complex scalar. This shows that for any value of $\mathbf{y}$, $\mathbf{y}^{+}\mathbf{\Lambda}\mathbf{y} \geq 0$, and therefore $\mathbf{\Lambda}$ is positive semi-definite.

Our new problem is to minimize Eq. 3.63 subject to the constraints in Eq. 3.65. This is equivalent to maximizing $\mathbf{y}^{+}\mathbf{\Lambda}\mathbf{y}$ subject to $\mathbf{y}^{+}\mathbf{y} = 1$ and the constraints in Eq. 3.65. The solution for this problem is outlined in [45]. We may form the functional

$$\mathcal{L}_{ZAMACH}(\mathbf{y}, \lambda, \boldsymbol{\mu}) = \mathbf{y}^{+}\mathbf{\Lambda}\mathbf{y} - \lambda(\mathbf{y}^{+}\mathbf{y} - 1) + \sum_{i=1}^{p} 2\mu_i \mathbf{y}^{+}\mathbf{c}_i \tag{3.69}$$

where we can write the matrix $\mathbf{C}$ as

$$\mathbf{C} = \begin{bmatrix} \mathbf{c}_1 & \mathbf{c}_2 & \cdots & \mathbf{c}_p \end{bmatrix} \tag{3.70}$$

and the λ and $\boldsymbol{\mu}$ are Lagrange multipliers, with $\boldsymbol{\mu}$ given by

$$\boldsymbol{\mu} = \begin{bmatrix} \mu_1 & \mu_2 & \cdots & \mu_p \end{bmatrix}^{T} \tag{3.71}$$

The number of zero-aliasing constraints, p, is equal to $N^{\#} - N_x^{\#}$. We can take the gradient of

$\mathcal{L}_{ZAMACH}$ in Eq. 3.69 with respect to $\mathbf{y}$ and set it to zero,

$$2\mathbf{\Lambda y} - 2\lambda\mathbf{y} + 2\sum_{i=1}^{p}\mu_i\mathbf{c}_i = \mathbf{0}$$

$$\mathbf{\Lambda y} - \lambda\mathbf{y} + \mathbf{C}\boldsymbol{\mu} = \mathbf{0} \tag{3.72}$$

We now introduce $\tilde{\mathrm{C}}$, the generalized inverse of C, which is given by

$$\tilde{\mathbf{C}} = (\mathbf{C}^{+}\mathbf{C})^{-1}\mathbf{C}^{+} \tag{3.73}$$

Note that this implies that $\tilde{\mathbf{C}}\mathbf{C} = \mathbf{I}$. We may now multiply both sides of Eq. 3.72 by $\tilde{\mathrm{C}}$ to obtain

$$\tilde{\mathbf{C}}\mathbf{\Lambda y} - \lambda\tilde{\mathbf{C}}\mathbf{y} + \tilde{\mathbf{C}}\mathbf{C}\boldsymbol{\mu} = \mathbf{0}$$

$$\tilde{\mathbf{C}}\mathbf{\Lambda y} - \lambda(\mathbf{C}^{+}\mathbf{C})^{-1}\mathbf{C}^{+}\mathbf{y} + \boldsymbol{\mu} = \mathbf{0}$$

$$\boldsymbol{\mu} = -\tilde{\mathbf{C}}\mathbf{\Lambda y} \tag{3.74}$$

Where we have used the fact that $\mathbf{C}^{+}\mathbf{y} = \mathbf{0}$. We may now substitute this result into Eq. 3.72 to obtain

$$\mathbf{\Lambda y} - \lambda\mathbf{y} - \mathbf{C}\tilde{\mathbf{C}}\mathbf{\Lambda y} = \mathbf{0}$$

$$(\mathbf{I} - \mathbf{C}\tilde{\mathbf{C}})\mathbf{\Lambda y} = \lambda\mathbf{y}$$

$$\mathbf{P}\mathbf{\Lambda y} = \lambda\mathbf{y} \tag{3.75}$$

where

$$\mathbf{P} = \mathbf{I} - \mathbf{C}\tilde{\mathbf{C}} \tag{3.76}$$

This is simply an eigenvalue problem. To get a closed form solution for the ZAMACH filter, we may take Eq. 3.75 and substitute for $\mathbf{\Lambda}$,

$$\mathbf{P}\mathbf{T}^{-1/2}\bar{\mathbf{m}}\bar{\mathbf{m}}^{+}\mathbf{T}^{-1/2}\mathbf{y} = \lambda\mathbf{y} \tag{3.77}$$

Because $\bar{\mathbf{m}}\bar{\mathbf{m}}^{+}$ is of rank 1, there is only one eigenvalue. We observe that $\alpha = \bar{\mathbf{m}}^{+}\mathbf{T}^{-1/2}\mathbf{y}$ is simply a constant. Therefore we can rewrite Eq. 3.77 as

$$\alpha\mathbf{P}\mathbf{T}^{-1/2}\bar{\mathbf{m}} = \lambda\mathbf{y} \tag{3.78}$$

We now substitute for $\mathbf{y}$, and solve for $\bar{\mathbf{h}}$,

$$\begin{aligned}\alpha\mathbf{P}\mathbf{T}^{-1/2}\bar{\mathbf{m}} &= \lambda\mathbf{T}^{1/2}\bar{\mathbf{h}} \\ \bar{\mathbf{h}} &= \frac{\alpha}{\lambda}\mathbf{T}^{-1/2}\mathbf{P}\mathbf{T}^{-1/2}\bar{\mathbf{m}}\end{aligned} \tag{3.79}$$

We now substitute for $\mathbf{P}$ and subsequently write $\tilde{\mathbf{C}}$ in terms of $\mathbf{C}$ to obtain

$$\begin{aligned}\bar{\mathbf{h}} &= \frac{\alpha}{\lambda}\mathbf{T}^{-1/2}(\mathbf{I} - \mathbf{C}\tilde{\mathbf{C}})\mathbf{T}^{-1/2}\bar{\mathbf{m}} \\ &= \frac{\alpha}{\lambda}\mathbf{T}^{-1/2}(\mathbf{I} - \mathbf{C}(\mathbf{C}^{+}\mathbf{C})^{-1}\mathbf{C}^{+})\mathbf{T}^{-1/2}\bar{\mathbf{m}} \\ &= \frac{\alpha}{\lambda}\left[\mathbf{T}^{-1/2}\mathbf{I}\mathbf{T}^{-1/2}\bar{\mathbf{m}} - \mathbf{T}^{-1/2}\mathbf{C}(\mathbf{C}^{+}\mathbf{C})^{-1}\mathbf{C}^{+}\mathbf{T}^{-1/2}\bar{\mathbf{m}}\right]\end{aligned} \tag{3.80}$$

We now substitute for $\mathbf{C}$ and simplify,

$$\begin{aligned}\bar{\mathbf{h}} &= \frac{\alpha}{\lambda}\left[\mathbf{T}^{-1}\bar{\mathbf{m}} - \mathbf{T}^{-1/2}\mathbf{T}^{-1/2}\mathbf{A}(\mathbf{A}^{+}\mathbf{T}^{-1/2}\mathbf{T}^{-1/2}\mathbf{A})^{-1}\mathbf{A}^{+}\mathbf{T}^{-1/2}\mathbf{T}^{-1/2}\bar{\mathbf{m}}\right] \\ &= \frac{\alpha}{\lambda}\left[\mathbf{T}^{-1}\bar{\mathbf{m}} - \mathbf{T}^{-1}\mathbf{A}(\mathbf{A}^{+}\mathbf{T}^{-1}\mathbf{A})^{-1}\mathbf{A}^{+}\mathbf{T}^{-1}\bar{\mathbf{m}}\right] \\ &= \frac{\alpha}{\lambda}\mathbf{\Delta}_{\mathbf{T}}\mathbf{T}^{-1}\bar{\mathbf{m}} \end{aligned} \tag{3.81}$$

When $\bar{\mathbf{p}} = \bar{\mathbf{m}}$ (i.e., the desired correlation output is a delta function), note that Eq. 3.81 matches that of 3.46 by a scale factor. This scale factor has no significance to the filter. Therefore we have shown that the ZAMACH filter, which explicitly maximizes average correlation height, is still adequately described by the ZAUCF filter, which maintains its role as a generalized formulation for unconstrained ZACFs.

3.6 ASEF and MOSSE

In this section, we discuss the ASEF [20] and MOSSE [21] filters.

3.6.1 ASEF

The ASEF filter as presented in [20] does not actually optimize any cost function. Instead, it computes what are denoted "exact filters" for each training image. These exact filters are then averaged to obtain the ASEF filter. The exact filters are given by

$$H_q(k,l) = \frac{G_{D,q}(k,l)}{X_q^*(k,l)} \tag{3.82}$$

where $G_{D,q}$ is the 2D DFT of the desired correlation plane, $X_q(k,l)$ is the 2D DFT of training image q, and $H_q(k,l)$ is the exact filter for training image q. The ASEF filter is then obtained by averaging the exact filters,

$$H(k,l) = \frac{1}{Q}\sum_{q=1}^{Q} H_q(k,l) \tag{3.83}$$

Note that the BSCF [22] is very similar to this formulation, with the exception that the weights for each of the exact filters are not equal. Before we can present a zero-aliasing version of the ASEF filter, we must determine what optimization metric ASEF corresponds to. After all, the expression in 3.82 implicitly assumes a circular correlation.

We can rewrite Eq. 3.82 in the matrix-vector notation as

$$\bar{\mathbf{h}}_q = (\bar{\mathbf{X}}_q^+)^{-1}\bar{\mathbf{g}}_{D,q} \tag{3.84}$$

with the exact filter given by

$$\bar{\mathbf{h}} = \frac{1}{Q}\sum_{q=1}^{Q} \bar{\mathbf{h}}_q \tag{3.85}$$

We also assert that each exact filter in the ASEF formulation minimizes the cost function

$$J(\bar{\mathbf{h}}_q) = \left\|\bar{\mathbf{X}}_q^+\bar{\mathbf{h}}_q - \bar{\mathbf{g}}_{D,q}\right\|^2 \tag{3.86}$$

with respect to $\bar{\mathbf{h}}_q$. It should be noted that this is never actually mentioned in [20]. To show this, we expand Eq. 3.86,

$$\begin{aligned} J(\bar{\mathbf{h}}_q) &= (\bar{\mathbf{X}}_q^+\bar{\mathbf{h}}_q - \bar{\mathbf{g}}_{D,q})^+(\bar{\mathbf{X}}_q^+\bar{\mathbf{h}}_q - \bar{\mathbf{g}}_{D,q}) \\ &= \bar{\mathbf{h}}_q^+\bar{\mathbf{X}}_q\bar{\mathbf{X}}_q^+\bar{\mathbf{h}}_q - \bar{\mathbf{h}}_q^+\bar{\mathbf{X}}_q\bar{\mathbf{g}}_{D,q} - \bar{\mathbf{g}}_{D,q}^+\bar{\mathbf{X}}_q^+\bar{\mathbf{h}}_q + \bar{\mathbf{g}}_{D,q}^+\bar{\mathbf{g}}_{D,q} \end{aligned} \tag{3.87}$$

Differentiating with respect to $\bar{\mathbf{h}}_q$ and setting to zero we obtain

$$\bar{\mathbf{X}}_q\bar{\mathbf{X}}_q^+\bar{\mathbf{h}}_q - \bar{\mathbf{X}}_q\bar{\mathbf{g}}_{D,q} = \mathbf{0}$$
$$\bar{\mathbf{h}}_q = (\bar{\mathbf{X}}_q\bar{\mathbf{X}}_q^+)^{-1}\bar{\mathbf{X}}_q\bar{\mathbf{g}}_{D,q}$$
$$\bar{\mathbf{h}}_q = (\bar{\mathbf{X}}_q^+)^{-1}\bar{\mathbf{g}}_{D,q} \tag{3.88}$$

Note that this matches Eq. 3.84, which proves that each exact filter minimizes Eq. 3.86. Therefore, to derive the zero-aliasing form of ASEF, we may start with Eq. 3.86.

3.6.2 ZAASEF

The zero-aliasing ASEF (ZAASEF) filter is obtained in two steps. First, we must solve for the zero-aliasing exact filters; then, we must solve for the ZAASEF by simply averaging these exact filters. The zero-aliasing exact filters are obtained by minimizing Eq. 3.86 subject to $\mathbf{A}^+\bar{\mathbf{h}_q} = \mathbf{0}$. We form the functional

$$\mathcal{L}_{ZAASEF}(\bar{\mathbf{h}}, \boldsymbol{\omega}) = \left\|\bar{\mathbf{X}}_q^+\bar{\mathbf{h}}_q - \bar{\mathbf{g}}_{D,q}\right\|^2 - \boldsymbol{\omega}^+\mathbf{A}^+\bar{\mathbf{h}}_q \tag{3.89}$$

Differentiating with respect to $\bar{\mathbf{h}}_q$ and setting to zero we obtain

$$\bar{\mathbf{X}}_q\bar{\mathbf{X}}_q^+\bar{\mathbf{h}}_q - \bar{\mathbf{X}}_q\bar{\mathbf{g}}_{D,q} - \mathbf{A}\boldsymbol{\omega} = \mathbf{0}$$
$$\bar{\mathbf{h}}_q = (\bar{\mathbf{X}}_q\bar{\mathbf{X}}_q^+)^{-1}(\bar{\mathbf{X}}_q\bar{\mathbf{g}}_{D,q} + \mathbf{A}\boldsymbol{\omega})$$
$$= \mathbf{D}_q^{-1}(\bar{\mathbf{X}}_q\bar{\mathbf{g}}_{D,q} + \mathbf{A}\boldsymbol{\omega}) \tag{3.90}$$

Where $\mathbf{D}_q = \bar{\mathbf{X}}_q\bar{\mathbf{X}}_q^+$. Substituting this into the zero-aliasing constraints, we obtain

$$\mathbf{A}^{+}\mathbf{D}_q^{-1}(\bar{\mathbf{X}}_q\bar{\mathbf{g}}_{D,q} + \mathbf{A}\boldsymbol{\omega}) = \mathbf{0}$$

$$\mathbf{A}^{+}\mathbf{D}_q^{-1}\bar{\mathbf{X}}_q\bar{\mathbf{g}}_{D,q} + \mathbf{A}^{+}\mathbf{D}_q^{-1}\mathbf{A}\boldsymbol{\omega} = \mathbf{0}$$

$$\boldsymbol{\omega} = -(\mathbf{A}^{+}\mathbf{D}_q^{-1}\mathbf{A})^{-1}\mathbf{A}^{+}\mathbf{D}_q^{-1}\bar{\mathbf{X}}_q\bar{\mathbf{g}}_{D,q} \tag{3.91}$$

Substituting this back into Eq. 3.90, we obtain the zero-aliasing exact filter for training image q as

$$\begin{aligned}\bar{\mathbf{h}}_q &= \mathbf{D}_q^{-1}(\bar{\mathbf{X}}_q\bar{\mathbf{g}}_{D,q} - \mathbf{A}(\mathbf{A}^{+}\mathbf{D}_q^{-1}\mathbf{A})^{-1}\mathbf{A}^{+}\mathbf{D}_q^{-1}\bar{\mathbf{X}}_q\bar{\mathbf{g}}_{D,q}) \\ &= \boldsymbol{\Delta}_{\mathbf{D}_q}\mathbf{D}_q^{-1}\bar{\mathbf{X}}_q\bar{\mathbf{g}}_{D,q}\end{aligned} \tag{3.92}$$

where

$$\boldsymbol{\Delta}_{\mathbf{D}_q} = \mathbf{I} - \mathbf{D}_q^{-1}\mathbf{A}(\mathbf{A}^{+}\mathbf{D}_q^{-1}\mathbf{A})^{-1}\mathbf{A}^{+} \tag{3.93}$$

The ZAASEF is then given as the average of the zero-aliasing exact filters,

$$\bar{\mathbf{h}} = \frac{1}{Q}\sum_{q=1}^{Q}\bar{\mathbf{h}}_q \tag{3.94}$$

where $\bar{\mathbf{h}}_q$ is given in Eq. 3.92.

3.6.3 MOSSE

One disadvantage of the ASEF filter is that it requires a large number of training images to generate a filter that performs well. The MOSSE filter [21] was proposed as a solution to this. MOSSE is an unconstrained filter that minimizes the MSE between the correlation of the CF template with the training signal(s) and the desired correlation output. For example, the desired

correlation output in the time domain could take on a value of one at the location of a positive class signal and zeros elsewhere. The MOSSE filter with a desired output of a delta function is essentially a MACH filter. The MOSSE filter may be solved by minimizing the function

$$J(\bar{\mathbf{h}}) = \frac{1}{Q}\sum_{q=1}^{Q}\left\|\bar{\mathbf{X}}_q^{+}\bar{\mathbf{h}} - \bar{\mathbf{g}}_{D,q}\right\|^2 \tag{3.95}$$

where $\bar{\mathbf{g}}_{D,q}$ represents the vectorized 2D DFT of the desired correlation output for the qth training image. Note the the product $\bar{\mathbf{X}}_q^{+}\bar{\mathbf{h}}$ represents the vectorized 2D DFT of the correlation of training image q and the template $h(n,m)$. However, this term is actually a circular correlation! Differentiating $J(\bar{\mathbf{h}})$ with respect to $\bar{\mathbf{h}}$ and setting to zero yields

$$\bar{\mathbf{h}} = \mathbf{D}^{-1}\bar{\mathbf{p}} \tag{3.96}$$

where $\mathbf{D}$ and p are given in Eq. 3.29 and 3.30, respectively (note that these expressions are off by a constant, but this is inconsequential to minimizing Eq. 3.95). The expression we have included here is written in matrix-vector notation, but is equivalent to the expression given in [21].

3.6.4 ZAMOSSE

Here, we introduce the zero-aliasing MOSSE (ZAMOSSE) filter. To remove the effects of circular correlation, we need to minimize Eq. 3.95 subject to the zero-aliasing constraints $\mathbf{A}^{+}\bar{\mathbf{h}} = \mathbf{0}$. We do this by forming the new functional

$$\mathcal{L}_{ZAMOSSE}(\bar{\mathbf{h}}, \boldsymbol{\omega}) = \frac{1}{Q}\sum_{q=1}^{Q}\left\|\bar{\mathbf{X}}_q^{+}\bar{\mathbf{h}} - \bar{\mathbf{g}}_{D,q}\right\|^2 - \boldsymbol{\omega}^{+}\mathbf{A}^{+}\bar{\mathbf{h}} \tag{3.97}$$

where $\boldsymbol{\omega}$ is a column vector of Lagrangian multipliers. Taking the gradient of $\mathcal{L}_{ZAMOSSE}(\bar{\mathbf{h}}, \boldsymbol{\omega})$

in Eq. 3.97 with respect to $\bar{\mathbf{h}}$ and setting it to zero, we obtain

$$\bar{\mathbf{h}} = \mathbf{D}^{-1}(\bar{\mathbf{p}} + \mathbf{A}\boldsymbol{\omega}) \tag{3.98}$$

We then substitute this back into the zero-aliasing constraints and solve for $\boldsymbol{\omega}$,

$$\boldsymbol{\omega} = -(\mathbf{A}^{+}\mathbf{D}^{-1}\mathbf{A})^{-1}\mathbf{A}^{+}\mathbf{D}^{-1}\bar{\mathbf{p}} \tag{3.99}$$

Substituting this back into Eq. 3.98, we obtain the ZAMOSSE filter,

$$\begin{aligned} \bar{\mathbf{h}} &= \mathbf{D}^{-1}\bar{\mathbf{p}} - \mathbf{D}^{-1}\mathbf{A}(\mathbf{A}^{+}\mathbf{D}^{-1}\mathbf{A})^{-1}\mathbf{A}^{+}\mathbf{D}^{-1}\bar{\mathbf{p}} \\ &= \boldsymbol{\Delta}_{\mathbf{D}}\mathbf{D}^{-1}\bar{\mathbf{p}} \end{aligned} \tag{3.100}$$

where

$$\boldsymbol{\Delta}_{\mathbf{D}} = \mathbf{I} - \mathbf{D}^{-1}\mathbf{A}(\mathbf{A}^{+}\mathbf{D}^{-1}\mathbf{A})^{-1}\mathbf{A}^{+} \tag{3.101}$$

In our MOSSE and ZAMOSSE filter implementations, we replace $\mathbf{D}$ with $\mathbf{T} = \lambda\mathbf{D} + (1 - \lambda\mathbf{P})$ as was described in the OTSDF formulation. This is referred to as regularization in [21], and is commonly used in optimal tradeoff filters (OTSDF, UOTSDF, etc.) to provide better noise tolerance [14, 16, 17, 46].

3.7 MMCF

In [23, 47], the MMCF was introduced. This filter combines the ideas of correlation filters with that of SVMs by removing the hard peak constraints and replacing them with inequality constraints. We first introduce the general idea of an SVM before introducing the MMCF filter.

3.7.1 Introduction to SVM

Given a set of training signals, the SVM approach finds a vector $\mathbf{h}$ (in the space domain) that maximizes the smallest L2 norm distance between a training sample and a hyperplane (which is defined by the vector $\mathbf{h}$). This can be expressed as

$$\max_{\mathbf{h},b'} \left(\min_{q} \left(\frac{l_q(\mathbf{h}^T\mathbf{x}_q + b')}{|\mathbf{h}|} \right) \right) \tag{3.102}$$

Note that, $\mathbf{x}_q$ is the qth (vectorized) training signal in the space domain (of which there are Q total), and $\mathbf{h}$ is a space domain vector. The variable b' here is called the bias term, and l_q is the class label (typically -1 or 1) of training signal $\mathbf{x}_q$. The term $\mathbf{h}^T\mathbf{x}_q + b'$ represents the projection of training signal $\mathbf{x}_q$ onto a hyperplane. This formulation does not give a unique solution - that is, any solution $(\mathbf{h}, b')$ will have infinitely many solutions $(\alpha\mathbf{h}, \alpha b')$, where α is a scalar. One common idea is to therefore constrain the term $l_q(\mathbf{h}^T\mathbf{x}_q + b')$ to unity for the signal closest to the hyperplane, i.e.

$$l_q(\mathbf{h}^T\mathbf{x}_q + b') = 1 \tag{3.103}$$

where q represents the training signal closest to the boundary. This can also be expressed as

$$\min_{q}(l_q(\mathbf{h}^T\mathbf{x}_q + b')) = 1 \tag{3.104}$$

An equivalent statement is to say that the projections of all training signals on the hyperplane

yield a value greater than 1, that is

$$l_q(\mathbf{h}^T\mathbf{x}_q + b^{'}) \geq 1 \tag{3.105}$$

for all q. Therefore, the optimization problem in Eq. 3.102 can be rewritten as

$$\begin{aligned} &\max_{\mathbf{h},b'} \frac{1}{|\mathbf{h}|} \\ &= \min_{\mathbf{h},b'} \mathbf{h}^T\mathbf{h} \end{aligned} \tag{3.106}$$

subject to the constraints

$$l_q(\mathbf{h}^T\mathbf{x}_q + b^{'}) \geq l_q u_q^{'} \tag{3.107}$$

where we have simply generalized the bound to the product of the class label l_q and the peak value $u_q^{'}$. This so-called hard-margin SVM formulation assumes that all of the training signals are linearly separable and all fall on the correct side of the margin. However, this is not always the case in practice. Therefore, it is common to introduce a slack variable for each training signal, $\xi_q^{'}$, which allows for training signals to be on the wrong side of the margin (in other words, it allows some training signals to be misclassified). The idea here is that the obtained margin with slack variables may be more appropriate (and more generalized) to avoid an outlier training signal from skewing the classifier. The so-called soft margin SVM formulation relaxes the constraints. The new optimization is given by

$$\min_{\mathbf{h},b'} \mathbf{h}^T\mathbf{h} + 2C\sum_{q=1}^{Q} \xi_q^{'} \tag{3.108}$$

such that

$$l_q(\mathbf{h}^T\mathbf{x}_q + b^{'}) \geq l_q u_q^{'} - \xi_q^{'} \tag{3.109}$$

where $\xi_q^{'} \geq 0$ for all q. The variable C is a scalar that controls the "penalty" term that penalizes samples from being on the wrong side of the margin. This problem may be formulated in vector notation as

$$\min_{\mathbf{h},b} \mathbf{h}^T\mathbf{h} + 2C\mathbf{1}^T\boldsymbol{\xi}^{'} \tag{3.110}$$

such that

$$\mathbf{L}(\mathbf{X}^T\mathbf{h} + b^{'}\mathbf{1}) \geq \mathbf{L}\mathbf{u}^{'} - \boldsymbol{\xi}^{'} \tag{3.111}$$

where $\mathbf{L}$ is a $Q \times Q$ diagonal matrix with the class label values $l_1, \ldots, l_Q$ along the diagonal, $\mathbf{u}^{'}$ is a vector formed from the training signal peak constraints, $u_1^{'}, \ldots u_Q^{'}$, and the matrix $\mathbf{X}$ is formed by concatenating the vectors $\mathbf{x}_1, \ldots, \mathbf{x}_Q$. It can be shown that

$$\mathbf{h} = \mathbf{X}\mathbf{L}\mathbf{a} \tag{3.112}$$

where the vector $\mathbf{a}$ is solved numerically with the quadratic programming problem

$$\max_{\mathbf{0} \leq \mathbf{a} \leq C\mathbf{1}} 2\mathbf{a}^T\mathbf{L}\mathbf{u}^{'} - \mathbf{a}^T\mathbf{L}\mathbf{X}^T\mathbf{X}\mathbf{L}\mathbf{a} \tag{3.113}$$

such that $\mathbf{a}^T\mathbf{L}\mathbf{1} = 0$. This can be expressed in the frequency domain as

$$\max_{\mathbf{0}\leq\mathbf{a}\leq C\mathbf{1}} 2\mathbf{a}^T\mathbf{L}(\frac{1}{N}\mathbf{u}) - \mathbf{a}^T\mathbf{L}(\frac{1}{N}\bar{\mathbf{X}}^+\bar{\mathbf{X}})\mathbf{L}\mathbf{a}$$
$$\max_{\mathbf{0}\leq\mathbf{a}\leq C\mathbf{1}} \frac{1}{N\#}(2\mathbf{a}^T\mathbf{L}\mathbf{u} - \mathbf{a}^T\mathbf{L}\bar{\mathbf{X}}^+\bar{\mathbf{X}}\mathbf{L}\mathbf{a})$$
$$\max_{\mathbf{0}\leq\mathbf{a}\leq C\mathbf{1}} 2\mathbf{a}^T\mathbf{L}\mathbf{u} - \mathbf{a}^T\mathbf{L}\bar{\mathbf{X}}^+\bar{\mathbf{X}}\mathbf{L}\mathbf{a} \tag{3.114}$$

such that $\mathbf{a}^T\mathbf{L}\mathbf{1} = 0$. The columns of matrix $\bar{\mathbf{X}}$ consist of vectorized DFTs of training signals, $\bar{\mathbf{x}}_1, \ldots, \bar{\mathbf{x}}_Q$.Note that $\mathbf{X}^T\mathbf{X} = \frac{1}{N}\bar{\mathbf{X}}^+\bar{\mathbf{X}}$ (Parseval's Theorem). We have chosen $\mathbf{u} = N\mathbf{u}'$ for convenience (we use 1D notation for simplicity). In a similar manner, the primal problem may also be expressed in the frequency domain as

$$\min_{\mathbf{h},b} \frac{1}{N}\bar{\mathbf{h}}^+\bar{\mathbf{h}} + 2C\mathbf{1}^T\frac{1}{N}\boldsymbol{\xi}$$
$$\min_{\mathbf{h},b} \frac{1}{N}\left(\bar{\mathbf{h}}^+\bar{\mathbf{h}} + 2C\mathbf{1}^T\boldsymbol{\xi}\right)$$
$$\min_{\mathbf{h},b} \bar{\mathbf{h}}^+\bar{\mathbf{h}} + 2C\mathbf{1}^T\boldsymbol{\xi} \tag{3.115}$$

subject to the constraints

$$\mathbf{L}(\frac{1}{N}\bar{\mathbf{X}}^+\bar{\mathbf{h}} + \frac{1}{N}b\mathbf{1}) \geq \mathbf{L}(\frac{1}{N}\mathbf{u}) - \frac{1}{N}\boldsymbol{\xi}$$
$$\mathbf{L}(\bar{\mathbf{X}}^+\bar{\mathbf{h}} + b\mathbf{1}) \geq \mathbf{L}\mathbf{u} - \boldsymbol{\xi} \tag{3.116}$$

Note that $\bar{\mathbf{h}}$ represents the hyperplane in the frequency domain. From Parseval's Theorem, $\mathbf{h}^T\mathbf{h} = \frac{1}{N}\bar{\mathbf{h}}^+\bar{\mathbf{h}}$ and $\mathbf{X}^T\mathbf{h} = \frac{1}{N}\bar{\mathbf{X}}^+\bar{\mathbf{h}}$. We have made the substitution $\mathbf{u} = N\mathbf{u}'$, $b = Nb'$, and $\boldsymbol{\xi} = N\boldsymbol{\xi}'$ for convenience. We introduce this frequency domain notation as it will be used from now on.

3.7.2 MMCF

The MMCF combines ideas from correlation filters with the constraints from SVMs. The MMCF is formulated similar to the CCF filter, as

$$\min_{\bar{\mathbf{h}},b} \bar{\mathbf{h}}^{+}\mathbf{T}\bar{\mathbf{h}} - 2\bar{\mathbf{h}}^{+}\bar{\mathbf{p}} + 2C\mathbf{1}^{T}\boldsymbol{\xi} \tag{3.117}$$

subject to the constraints

$$\mathbf{L}(\bar{\mathbf{X}}^{+}\bar{\mathbf{h}} + b\mathbf{1}) \geq \mathbf{L}\mathbf{u} - \boldsymbol{\xi} \tag{3.118}$$

$$\boldsymbol{\xi} \geq \mathbf{0} \tag{3.119}$$

where $\boldsymbol{\xi} = [\xi_1, \ldots, \xi_Q]^T$ is a vector of slack variables that penalize features that are on the wrong side of the margin and C is a tradeoff parameter. The other quantities are the same as in the CCF formulation, and are given by $\mathbf{T} = (1-\gamma)\mathbf{D} - \gamma\mathbf{M} + \beta\mathbf{P}$, $\bar{\mathbf{p}} = \frac{1}{NQ}\sum_{q=1}^{Q}\bar{\mathbf{X}}_q\bar{\mathbf{g}}_{D,q}$, and $\mathbf{u} = [u_1, \ldots, u_Q]^T$. As shown in [47], the filter may be solved as

$$\bar{\mathbf{h}} = \mathbf{T}^{-1}\bar{\mathbf{p}} + \mathbf{T}^{-1}\bar{\mathbf{X}}\mathbf{L}\mathbf{a} \tag{3.120}$$

where the vector $\mathbf{a}$ is solved numerically with the quadratic programming problem

$$\max_{\mathbf{0}\leq\mathbf{a}\leq C\mathbf{1}} \mathbf{a}^{T}\mathbf{M}\mathbf{a} + \mathbf{a}^{T}\mathbf{b} \tag{3.121}$$

where

$$\mathbf{M} = -\mathbf{L}\bar{\mathbf{X}}^{+}\mathbf{T}^{-1}\bar{\mathbf{X}}\mathbf{L} \tag{3.122}$$

$$\mathbf{b} = 2\mathbf{L}(\mathbf{u} - \bar{\mathbf{X}}^{+}\mathbf{T}^{-1}\bar{\mathbf{p}}) \tag{3.123}$$

subject to the constraints $\mathbf{a}^T\mathbf{L}\mathbf{1} = 0$, which may be written as

$$\mathbf{a}^T\mathbf{y} = 0 \tag{3.124}$$

where $\mathbf{y}$ is a vector formed from the diagonal of $\mathbf{L}$,

$$\mathbf{y} = diag(\mathbf{L}) \tag{3.125}$$

The details of this derivation may be found in [47].

3.7.3 ZAMMCF

In this section, we detail the derivation of the zero-aliasing MMCF (ZAMMCF) filter, which adds zero-aliasing constraints to the MMCF formulation. The ZAMMCF is formulated as

$$\min_{\bar{\mathbf{h}},b} \bar{\mathbf{h}}^+\mathbf{T}\bar{\mathbf{h}} - 2\bar{\mathbf{h}}^+\bar{\mathbf{p}} + 2C\mathbf{1}^T\boldsymbol{\xi} \tag{3.126}$$

subject to the constraints

$$\mathbf{L}(\bar{\mathbf{X}}^+\bar{\mathbf{h}} + b\mathbf{1}) \geq \mathbf{L}\mathbf{u} - \boldsymbol{\xi} \tag{3.127}$$

$$\boldsymbol{\xi} \geq \mathbf{0} \tag{3.128}$$

$$\mathbf{A}^+\bar{\mathbf{h}} = 0 \tag{3.129}$$

We may form a functional with Lagrange multipliers,

$$\mathcal{L}_{ZAMMCF}(\bar{\mathbf{h}}, \boldsymbol{\xi}, b, \mathbf{a}, \boldsymbol{\mu}, \boldsymbol{\omega}) = \bar{\mathbf{h}}^+\mathbf{T}\bar{\mathbf{h}} - 2\bar{\mathbf{h}}^+\bar{\mathbf{p}} + 2C\mathbf{1}^T\boldsymbol{\xi} - 2\mathbf{a}^T[\mathbf{L}(\bar{\mathbf{X}}^+\bar{\mathbf{h}} + b\mathbf{1}) - \mathbf{L}\mathbf{u} + \boldsymbol{\xi}] - 2\boldsymbol{\mu}^T\boldsymbol{\xi} - 2\boldsymbol{\omega}^+\mathbf{A}^+\bar{\mathbf{h}} \tag{3.130}$$

where $\mathbf{a}, \boldsymbol{\mu} \geq \mathbf{0}$ and $\boldsymbol{\omega}$ are vectors of Lagrange multipliers. Taking the partial derivative with respect to $\bar{\mathbf{h}}$ and setting it equal to zero gives

$$\frac{\partial \mathcal{L}_{ZAMMCF}}{\partial \bar{\mathbf{h}}} = 2\mathbf{T}\bar{\mathbf{h}} - 2\bar{\mathbf{p}} - 2\bar{\mathbf{X}}\mathbf{L}\mathbf{a} - 2\mathbf{A}\boldsymbol{\omega} = \mathbf{0} \tag{3.131}$$

Solving for $\bar{\mathbf{h}}$ gives

$$\bar{\mathbf{h}} = \mathbf{T}^{-1}(\bar{\mathbf{p}} + \bar{\mathbf{X}}\mathbf{L}\mathbf{a} + \mathbf{A}\boldsymbol{\omega}) \tag{3.132}$$

Note that $\bar{\mathbf{h}}$ is a function of $\mathbf{a}$. The goal, therefore, is to rewrite Eq. 3.130 as a function of only $\mathbf{a}$. Then, we may optimize for $\mathbf{a}$ and therefore solve for $\bar{\mathbf{h}}$. Taking the partial derivatives of Eq. 3.130 with respect to b, $\boldsymbol{\xi}$, and $\boldsymbol{\omega}$, and setting them equal to zero gives

$$\frac{\partial \mathcal{L}_{ZAMMCF}}{\partial b} = -2\mathbf{a}^T\mathbf{L}\mathbf{1} = \mathbf{0} \tag{3.133}$$

$$\frac{\partial \mathcal{L}_{ZAMMCF}}{\partial \boldsymbol{\xi}} = 2C\mathbf{1} - 2\mathbf{a} - 2\boldsymbol{\mu} = \mathbf{0} \tag{3.134}$$

$$\frac{\partial \mathcal{L}_{ZAMMCF}}{\partial \boldsymbol{\omega}} = -2\mathbf{A}^+\bar{\mathbf{h}} = \mathbf{0} \tag{3.135}$$

If we solve Eq. 3.134 for $C\mathbf{1}$ gives

$$C\mathbf{1} = \mathbf{a} + \boldsymbol{\mu} \tag{3.136}$$

Given that $\mathbf{a}, \boldsymbol{\mu} \geq \mathbf{0}$ and $\mathbf{a} = C\mathbf{1} - \boldsymbol{\mu}$, then $\mathbf{0} \leq \mathbf{a} \leq C\mathbf{1}$. Also note that, from Eq. 3.133, $\mathbf{a}^T\mathbf{L}\mathbf{1} = \mathbf{0}$. These constraints will be used when solving for $\mathbf{a}$.

Next, we may substitute $\bar{\mathbf{h}}$ (Eq. 3.132) and $C\mathbf{1}$ (Eq. 3.136) into Eq. 3.130 to obtain

$$\mathcal{L}_{ZAMMCF}(\mathbf{a}) = \left[\mathbf{T}^{-1}(\bar{\mathbf{p}} + \bar{\mathbf{X}}\mathbf{L}\mathbf{a} + \mathbf{A}\boldsymbol{\omega})\right]^{+} \mathbf{T} \left[\mathbf{T}^{-1}(\bar{\mathbf{p}} + \bar{\mathbf{X}}\mathbf{L}\mathbf{a} + \mathbf{A}\boldsymbol{\omega})\right]$$
$$- 2\left[\mathbf{T}^{-1}(\bar{\mathbf{p}} + \bar{\mathbf{X}}\mathbf{L}\mathbf{a} + \mathbf{A}\boldsymbol{\omega})\right]^{+} \bar{\mathbf{p}} + 2(\mathbf{a} + \boldsymbol{\mu})^T\boldsymbol{\xi}$$
$$- 2\mathbf{a}^T[\mathbf{L}(\bar{\mathbf{X}}^{+}\mathbf{T}^{-1}(\bar{\mathbf{p}} + \bar{\mathbf{X}}\mathbf{L}\mathbf{a} + \mathbf{A}\boldsymbol{\omega}) + b\mathbf{1}) - \mathbf{L}\mathbf{u} + \boldsymbol{\xi}] - 2\boldsymbol{\mu}^T\boldsymbol{\xi} \quad (3.137)$$

where we have used the fact that $\mathbf{A}^{+}\bar{\mathbf{h}} = \mathbf{0}$ (Eq. 3.135). We may expand this expression to obtain

$$\mathcal{L}_{ZAMMCF}(\mathbf{a}) = (\bar{\mathbf{p}} + \bar{\mathbf{X}}\mathbf{L}\mathbf{a} + \mathbf{A}\boldsymbol{\omega})^{+}\mathbf{T}^{-1}(\bar{\mathbf{p}} + \bar{\mathbf{X}}\mathbf{L}\mathbf{a} + \mathbf{A}\boldsymbol{\omega})$$
$$- 2(\bar{\mathbf{p}} + \bar{\mathbf{X}}\mathbf{L}\mathbf{a} + \mathbf{A}\boldsymbol{\omega})^{+}\mathbf{T}^{-1}\bar{\mathbf{p}} + 2\mathbf{a}^T\boldsymbol{\xi} + 2\boldsymbol{\mu}^T\boldsymbol{\xi}$$
$$- 2\mathbf{a}^T\mathbf{L}\bar{\mathbf{X}}^{+}\mathbf{T}^{-1}(\bar{\mathbf{p}} + \bar{\mathbf{X}}\mathbf{L}\mathbf{a} + \mathbf{A}\boldsymbol{\omega}) - 2b\mathbf{a}^T\mathbf{L}\mathbf{1} + 2\mathbf{a}^T\mathbf{L}\mathbf{u} - 2\mathbf{a}^T\boldsymbol{\xi} - 2\boldsymbol{\mu}^T\boldsymbol{\xi} \quad (3.138)$$

We may simplify this expression and invoke Eq. 3.133 to obtain

$$\mathcal{L}_{ZAMMCF}(\mathbf{a}) = (\bar{\mathbf{p}}^{+} + \mathbf{a}^T\mathbf{L}\bar{\mathbf{X}}^{+} + \boldsymbol{\omega}^{+}\mathbf{A}^{+})\mathbf{T}^{-1}(\bar{\mathbf{p}} + \bar{\mathbf{X}}\mathbf{L}\mathbf{a} + \mathbf{A}\boldsymbol{\omega})$$
$$- 2(\bar{\mathbf{p}}^{+} + \mathbf{a}^T\mathbf{L}\bar{\mathbf{X}}^{+} + \boldsymbol{\omega}^{+}\mathbf{A}^{+})\mathbf{T}^{-1}\bar{\mathbf{p}}$$
$$- 2\mathbf{a}^T\mathbf{L}\bar{\mathbf{X}}^{+}\mathbf{T}^{-1}(\bar{\mathbf{p}} + \bar{\mathbf{X}}\mathbf{L}\mathbf{a} + \mathbf{A}\boldsymbol{\omega}) + 2\mathbf{a}^T\mathbf{L}\mathbf{u} \quad (3.139)$$

This may be further expanded to yield

$$\begin{aligned}\mathcal{L}_{ZAMMCF}(\mathbf{a}) = \bar{\mathbf{p}}^{+}\mathbf{T}^{-1}\bar{\mathbf{p}} + \mathbf{a}^{T}\mathbf{L}\bar{\mathbf{X}}^{+}\mathbf{T}^{-1}\bar{\mathbf{X}}\mathbf{L}\mathbf{a} + \boldsymbol{\omega}^{+}\mathbf{A}^{+}\mathbf{T}^{-1}\mathbf{A}\boldsymbol{\omega} \\ + 2\mathbf{a}^{T}\mathbf{L}\bar{\mathbf{X}}^{+}\mathbf{T}^{-1}\bar{\mathbf{p}} + 2\boldsymbol{\omega}^{+}\mathbf{A}^{+}\mathbf{T}^{-1}\bar{\mathbf{p}} + 2\mathbf{a}^{T}\mathbf{L}\bar{\mathbf{X}}^{+}\mathbf{T}^{-1}\mathbf{A}\boldsymbol{\omega} \\ - 2\bar{\mathbf{p}}^{+}\mathbf{T}^{-1}\bar{\mathbf{p}} - 2\mathbf{a}^{T}\mathbf{L}\bar{\mathbf{X}}^{+}\mathbf{T}^{-1}\bar{\mathbf{p}} - 2\boldsymbol{\omega}^{+}\mathbf{A}^{+}\mathbf{T}^{-1}\bar{\mathbf{p}} \\ - 2\mathbf{a}^{T}\mathbf{L}\bar{\mathbf{X}}^{+}\mathbf{T}^{-1}(\bar{\mathbf{p}} + \bar{\mathbf{X}}\mathbf{L}\mathbf{a} + \mathbf{A}\boldsymbol{\omega}) + 2\mathbf{a}^{T}\mathbf{L}\mathbf{u} \quad (3.140)\end{aligned}$$

Finally, we simplify to obtain

$$\mathcal{L}_{ZAMMCF}(\mathbf{a}) = -\mathbf{a}^{T}\mathbf{L}\bar{\mathbf{X}}^{+}\mathbf{T}^{-1}\bar{\mathbf{X}}\mathbf{L}\mathbf{a} + 2\mathbf{a}^{T}(\mathbf{L}\mathbf{u} - \mathbf{L}\bar{\mathbf{X}}^{+}\mathbf{T}^{-1}\bar{\mathbf{p}}) + \boldsymbol{\omega}^{+}\mathbf{A}^{+}\mathbf{T}^{-1}\mathbf{A}\boldsymbol{\omega} + \kappa \quad (3.141)$$

where

$$\kappa = -\bar{\mathbf{p}}^{+}\mathbf{T}^{-1}\bar{\mathbf{p}} \quad (3.142)$$

is a constant that does not affect the cost function (as it is not a function of $\mathbf{a}$). Note that Eq. 3.141 is a function of $\mathbf{a}$ and $\boldsymbol{\omega}$. We may substitute Eq. 3.132 into the constraints in Eq. 3.129, to obtain

$$\begin{aligned}\mathbf{A}^{+}\mathbf{T}^{-1}(\bar{\mathbf{p}} + \bar{\mathbf{X}}\mathbf{L}\mathbf{a} + \mathbf{A}\boldsymbol{\omega}) = \mathbf{0} \\ \mathbf{A}^{+}\mathbf{T}^{-1}\bar{\mathbf{p}} + \mathbf{A}^{+}\mathbf{T}^{-1}\bar{\mathbf{X}}\mathbf{L}\mathbf{a} + \mathbf{A}^{+}\mathbf{T}^{-1}\mathbf{A}\boldsymbol{\omega} = \mathbf{0} \\ \boldsymbol{\omega} = -(\mathbf{A}^{+}\mathbf{T}^{-1}\mathbf{A})^{-1}(\mathbf{A}^{+}\mathbf{T}^{-1}\bar{\mathbf{p}} + \mathbf{A}^{+}\mathbf{T}^{-1}\bar{\mathbf{X}}\mathbf{L}\mathbf{a}) \\ \boldsymbol{\omega} = -\boldsymbol{\Gamma}^{-1}\mathbf{A}^{+}(\mathbf{T}^{-1}\bar{\mathbf{p}} + \mathbf{T}^{-1}\bar{\mathbf{X}}\mathbf{L}\mathbf{a}) \quad (3.143)\end{aligned}$$

where we use the variable $\boldsymbol{\Gamma}$ for notational convenience,

$$\mathbf{\Gamma} = \mathbf{A}^{+}\mathbf{T}^{-1}\mathbf{A} \tag{3.144}$$

Substituting $\boldsymbol{\omega}$ into Eq. 3.141, we obtain

$$\begin{aligned}\mathcal{L}_{ZAMMCF}(\mathbf{a}) &= \kappa - \mathbf{a}^{T}\mathbf{L}\bar{\mathbf{X}}^{+}\mathbf{T}^{-1}\bar{\mathbf{X}}\mathbf{L}\mathbf{a} + 2\mathbf{a}^{T}(\mathbf{L}\mathbf{u} - \mathbf{L}\bar{\mathbf{X}}^{+}\mathbf{T}^{-1}\bar{\mathbf{p}}) \\ &\quad + \left[\mathbf{\Gamma}^{-1}(\mathbf{A}^{+}\mathbf{T}^{-1}\bar{\mathbf{p}} + \mathbf{A}^{+}\mathbf{T}^{-1}\bar{\mathbf{X}}\mathbf{L}\mathbf{a})\right]^{+}\mathbf{\Gamma}\left[\mathbf{\Gamma}^{-1}(\mathbf{A}^{+}\mathbf{T}^{-1}\bar{\mathbf{p}} + \mathbf{A}^{+}\mathbf{T}^{-1}\bar{\mathbf{X}}\mathbf{L}\mathbf{a})\right] \end{aligned} \tag{3.145}$$

$$\begin{aligned}&= \kappa - \mathbf{a}^{T}\mathbf{L}\bar{\mathbf{X}}^{+}\mathbf{T}^{-1}\bar{\mathbf{X}}\mathbf{L}\mathbf{a} + 2\mathbf{a}^{T}(\mathbf{L}\mathbf{u} - \mathbf{L}\bar{\mathbf{X}}^{+}\mathbf{T}^{-1}\bar{\mathbf{p}}) \\ &\quad + (\bar{\mathbf{p}}^{+}\mathbf{T}^{-1}\mathbf{A} + \mathbf{a}^{T}\mathbf{L}\bar{\mathbf{X}}^{+}\mathbf{T}^{-1}\mathbf{A})(\mathbf{\Gamma}^{+})^{-1}(\mathbf{A}^{+}\mathbf{T}^{-1}\bar{\mathbf{p}} + \mathbf{A}^{+}\mathbf{T}^{-1}\bar{\mathbf{X}}\mathbf{L}\mathbf{a}) \end{aligned} \tag{3.146}$$

Note that $\mathbf{\Gamma}^{+} = \mathbf{\Gamma}$,

$$\begin{aligned}\mathcal{L}_{ZAMMCF}(\mathbf{a}) &= \kappa - \mathbf{a}^{T}\mathbf{L}\bar{\mathbf{X}}^{+}\mathbf{T}^{-1}\bar{\mathbf{X}}\mathbf{L}\mathbf{a} + 2\mathbf{a}^{T}(\mathbf{L}\mathbf{u} - \mathbf{L}\bar{\mathbf{X}}^{+}\mathbf{T}^{-1}\bar{\mathbf{p}}) \\ &\quad + \kappa_2 + 2\mathbf{a}^{T}\mathbf{L}\bar{\mathbf{X}}^{+}\mathbf{T}^{-1}\mathbf{A}\mathbf{\Gamma}^{-1}\mathbf{A}^{+}\mathbf{T}^{-1}\bar{\mathbf{p}} + \mathbf{a}^{T}\mathbf{L}\bar{\mathbf{X}}^{+}\mathbf{T}^{-1}\mathbf{A}\mathbf{\Gamma}^{-1}\mathbf{A}^{+}\mathbf{T}^{-1}\bar{\mathbf{X}}\mathbf{L}\mathbf{a} \end{aligned} \tag{3.147}$$

$$\begin{aligned}&= \kappa + \kappa_2 - \mathbf{a}^{T}\mathbf{L}\bar{\mathbf{X}}^{+}\left[\mathbf{I} - \mathbf{T}^{-1}\mathbf{A}\mathbf{\Gamma}^{-1}\mathbf{A}^{+}\right]\mathbf{T}^{-1}\bar{\mathbf{X}}\mathbf{L}\mathbf{a}\ldots \\ &\quad + 2\mathbf{a}^{T}\mathbf{L}\left[\mathbf{u} - \bar{\mathbf{X}}^{+}(\mathbf{I} - \mathbf{T}^{-1}\mathbf{A}\mathbf{\Gamma}^{-1}\mathbf{A}^{+})\mathbf{T}^{-1}\bar{\mathbf{p}}\right] \end{aligned} \tag{3.148}$$

where

$$\kappa_2 = \bar{\mathbf{p}}^{+}\mathbf{T}^{-1}\mathbf{A}\mathbf{\Gamma}^{-1}\mathbf{A}^{+}\mathbf{T}^{-1}\bar{\mathbf{p}} \tag{3.149}$$

is a constant that does not affect the cost function (as it is not a function of $\mathbf{a}$). We may drop the constants write Eq. 3.148 as

$$\mathcal{L}_{ZAMMCF}(\mathbf{a}) = \mathbf{a}^T\hat{\mathbf{M}}\mathbf{a} + \mathbf{a}^T\hat{\mathbf{b}} \tag{3.150}$$

where

$$\hat{\mathbf{M}} = -\mathbf{L}\bar{\mathbf{X}}^+\mathbf{\Delta_T}\mathbf{T}^{-1}\bar{\mathbf{X}}\mathbf{L} \tag{3.151}$$

$$\hat{\mathbf{b}} = 2\mathbf{L}\left(\mathbf{u} - \bar{\mathbf{X}}^+\mathbf{\Delta_T}\mathbf{T}^{-1}\bar{\mathbf{p}}\right) \tag{3.152}$$

$$\mathbf{\Delta_T} = \mathbf{I} - \mathbf{T}^{-1}\mathbf{A}(\mathbf{A}^+\mathbf{T}^{-1}\mathbf{A})^{-1}\mathbf{A}^+ \tag{3.153}$$

To solve for the ZAMMCF filter, we first calculate $\mathbf{M}$ and $\mathbf{b}$. Then, we numerically solve the quadratic programming problem

$$\max_{\mathbf{0}\leq\mathbf{a}\leq C\mathbf{1}} \mathbf{a}^T\hat{\mathbf{M}}\mathbf{a} + \mathbf{a}^T\hat{\mathbf{b}} \tag{3.154}$$

subject to the constraints $\mathbf{a}^T\mathbf{L}\mathbf{1} = 0$, which may be written as

$$\mathbf{a}^T\mathbf{y} = 0 \tag{3.155}$$

where $\mathbf{y}$ is a vector formed from the diagonal of $\mathbf{L}$,

$$\mathbf{y} = diag(\mathbf{L}) \tag{3.156}$$

Finally, we first obtain $\boldsymbol{\omega}$ by substituting the vector $\mathbf{a}$ into Eq. 3.143. Then, we solve for the ZAMMCF filter by substituting the vectors $\mathbf{a}$ and $\boldsymbol{\omega}$ into Eq. 3.132.

NOTES: In this derivation, we maintained the bias term (b) in the ZAMMCF derivation. We have observed, however, that better performance is obtained by removing this bias term.

We speculate this is because the original MMCF formulation cost function expresses correlation without a bias term (the bias term only shows up in the constraints). It is possible better performance could be obtained by completely reformulating the MMCF objective function to include the bias term. However, this is beyond the scope of this thesis and is therefore reserved for future work. Note that if we remove the bias term from the above derivation, the changes are minimal. First, the constraint in Eq. 3.127 would instead be

$$\mathbf{L}\bar{\mathbf{X}}^{+}\bar{\mathbf{h}} \geq \mathbf{L}\mathbf{u} - \boldsymbol{\xi} \tag{3.157}$$

As a consequence, the term containing b (initially in Eq. 3.130) would disappear, and Eq. 3.133 would not exist. The answer is then the same as what was derived above, with the exception that the constraints $\mathbf{a}^T\mathbf{L}\mathbf{1} = 0$ on the numerical optimization in Eq. 3.154 would no longer exist.

3.8 Preliminary Results

In this section, we present initial results for ZACFs on test data. Here, we will only consider the task of face recognition on the AT&T/ORL face recognition dataset [43]. For computational reasons, we downsample each image to size 28×23 pixels. Recall that in the full dataset, there are 40 subjects with 10 images each. We normalize all images in the dataset but we do not subtract the mean[2] of each image (hereafter referred to zero-mean). For this dataset, our standard approach is to test using leave-one-out cross validation. For each experiment, 9 training images were used to train filters for each subject. These filters were then tested on the remaining image for all subjects. This resulted in a total of 40 test images (one from each class) that were tested with 40 filters each, for a total of 1600 correlations. This was repeated 10 times to capture every possible set of 9 training images. For each correlation, we calculate PCE (Eq. 2.1). Another score

[2]In general, we observe that aliasing effects are more severe when the images are not zero-mean. In general, better performance for both the original formulation and the ZACF formulation can be achieved when using zero-mean images. In the case of this experiment, however, using non-zero mean images aids in demonstrating the efficacy of the ZACF approach.

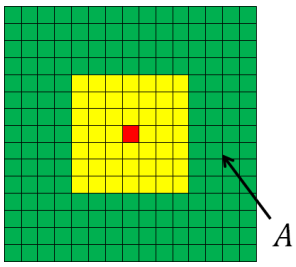


Figure 3.10: Illustration of the computation of PSR. We show a correlation peak (centered) in red. The region immediately around this peak, $\pm a$ pixels (shown in yellow) is set to 0. The annular region A around this (shown in green) has outer dimension $\pm b$ pixels away from the peak. This annular region is used to measure the properties of the peak's sidelobes. In this figure, $a = 3$ and $b = 7$.

metric is peak to sidelobe ratio (PSR), which gives a more localized measure of peak sharpness. Referring to Fig. 3.10, PSR is computed as [1]

$$PSR = \frac{c_{peak} - mean(A)}{std(A)} \tag{3.158}$$

where A denotes the area around the peak. In general we prefer PCE over PSR, but they give similar results in our experiments.

3.8.1 FDMACE vs. ZAMACE

In this experiment (see [41]), we build four filters for each subject:

- conventional FDMACE, using a DFT size of 28×23.
- conventional FDMACE, using zero padding on training images and a DFT size of 55×45, cropped to size 28×23.
- conventional FDMACE, using zero padding on training images and a DFT size of 55×45.
- ZAMACE, using zero padding on training images and a DFT size of 55×45, cropped to size 28×23.

The purpose of testing three types of FDMACE filters is to determine the effectiveness of padding the training images in different ways. Notice that the first two methods and the ZACF method

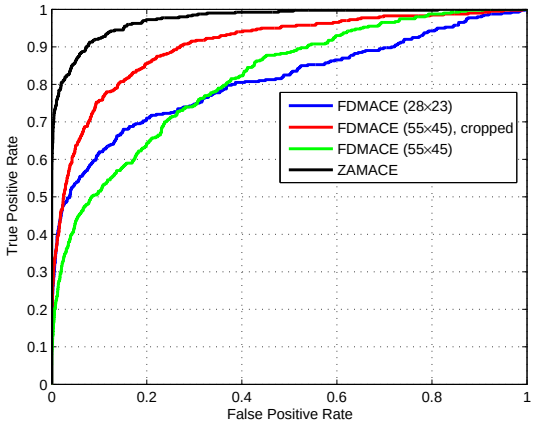


Figure 3.11: ROC curve for the AT&T/ORL dataset. Here the ROC curve represents 10 experiments, each of which uses a train size of 9 images. The score metric used here is PCE.

generate templates of size 28×23. When the full ZACF template is cropped, this is inconsequential, as the tail is zero already. The third method uses a full size DFT but does not crop, so the template is of size 55×45. The purpose of showing this method is to show that cropping this template to size 28×23 does not hurt performance.

We show an ROC curve in Fig. 3.11, using PCE as the score metric. We note that we get similar results for different training set sizes as well. We make the following observations:

- The padded and cropped FDMACE method outperforms the padded FDMACE, which indicates that cropping the full template helps;
- The padded and cropped FDMACE outperforms the FDMACE using the small DFT size (28×23). This is because some of the circular correlation effects are removed due to padding the images.
- The ZAMACE outperforms all other techniques by a substantial margin. This is because the ZAMACE eliminates all circular correlation effects during training, those due to the template (forcing the tail to zero) and those due to the training images (padding prior to taking the DFT).

Table 3.1: EER for Different Score Criteria, AT&T/ORL Dataset

Train Size = 3	PCE	PSR $(2, 6)$	PSR $(2, 10)$	PSR $(4, 10)$
FDMACE (28×23)	0.303	0.284	0.294	0.308
FDMACE (55×45)	0.279	0.255	0.270	0.279
FDMACE (55×45) cropped	0.195	0.238	0.229	0.232
ZAMACE	0.135	0.143	0.135	0.132
Train Size = 6				
FDMACE (28×23)	0.282	0.262	0.272	0.287
FDMACE (55×45)	0.278	0.259	0.270	0.282
FDMACE (55×45) cropped	0.184	0.237	0.227	0.234
ZAMACE	0.104	0.111	0.0975	0.0960
Train Size = 9				
FDMACE (28×23)	0.265	0.260	0.253	0.271
FDMACE (55×45)	0.270	0.260	0.270	0.280
FDMACE (55×45) cropped	0.170	0.236	0.223	0.235
ZAMACE	0.0856	0.0900	0.0850	0.0800

We further show the EER for all three filters in Table 3.1. We show these values for multiple training sizes as well as different scoring metrics. For each training size, we repeated the experiment 10 times with a different train and test set. For PSR, we denote the pair (a, b) to describe the size of the region as illustrated in Fig. 3.10. Based on these results, we move forward using PCE as our preferred score metric. We also typically will only show results for a train size of 9 on the AT&T/ORL dataset from this point forward.

3.8.2 Performance Insight

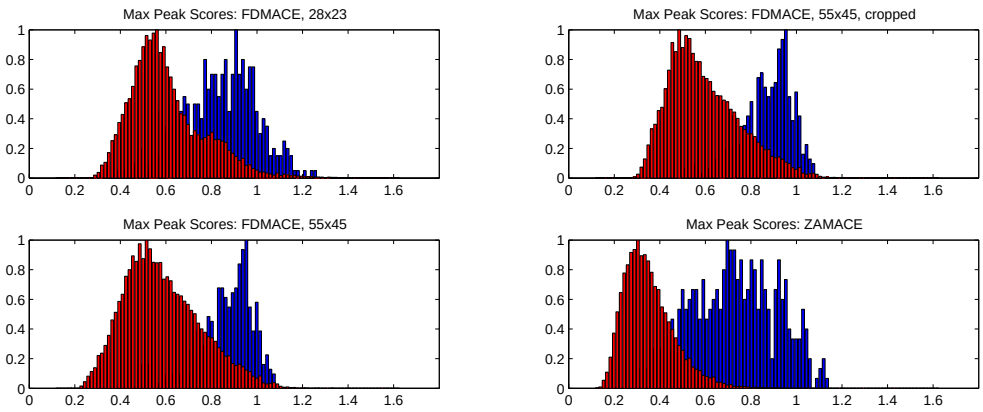
Here, we briefly investigate why the ZAMACE outperforms the conventional FDMACE. Fundamentally, ZAMACE offers a formulation that represents a linear correlation, whereas the conventional FDMACE formulation represents a circular correlation. Therefore, ZAMACE achieves better correlation outputs because the ACE is minimized more effectively than in the FDMACE case. This is always true for the training set, as has been shown earlier for both 1D and 2D examples. In general, ZAMACE achieves lower output correlation energy for test images as well

(although this is not guaranteed). This lower ACE yields a higher PCE score, which results in improved recognition performance. To illustrate this, we plot the distributions of the maximum peak value, correlation energy, and PCE scores for all of the test images (authentic and imposter) from the ORL experiment using a training size of 9. These distributions are shown in Fig. 3.12. In these plots, the distributions for the authentic cases are shown in blue and the distributions for imposter cases are shown in red. All histograms have been normalized to a maximum value of unity. Note that, in the test case, all four filters generally produce higher peaks for authentic classes than for imposter classes. However, ZAMACE yields significantly lower ACE for both imposter and authentic classes. As a result, the PCE for ZAMACE is considerably higher for authentic classes than for imposter classes. As a result, the ZAMACE is better able to differentiate between imposter and authentic classes, which directly leads to improved recognition performance.

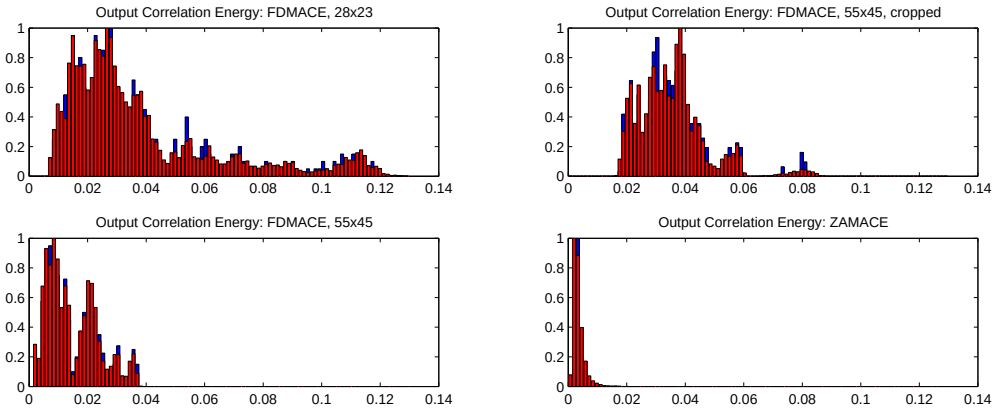
3.8.3 Results for Other CF Types

We repeated the previous experiment using other CF formulations. Here, we use as a baseline the padded and cropped FDMACE formulation, which we observed to be the best option for the original FDMACE formulation. We show the results of this evaluation in Table 3.2. In the table, we denote the filter type as well as the applicable parameters for the CF. For OTSDF, UOTSDF, and MMCF, the parameters reflect the choice of the tradeoff for $\mathbf{T} = \lambda\mathbf{D} + (1 - \lambda)\mathbf{P}$. For MSESDF, ASEF, and MOSSE, the σ^2 parameter denotes the variance of the desired correlation peak, which is Gaussian shape located at the origin of each training image. For the MACH filter, the parameters given are for the UCF formulation, which in turn correspond to the original MACH filter formulation [16] ($\mathbf{D}$ and $\mathbf{S}$ evenly weighted).

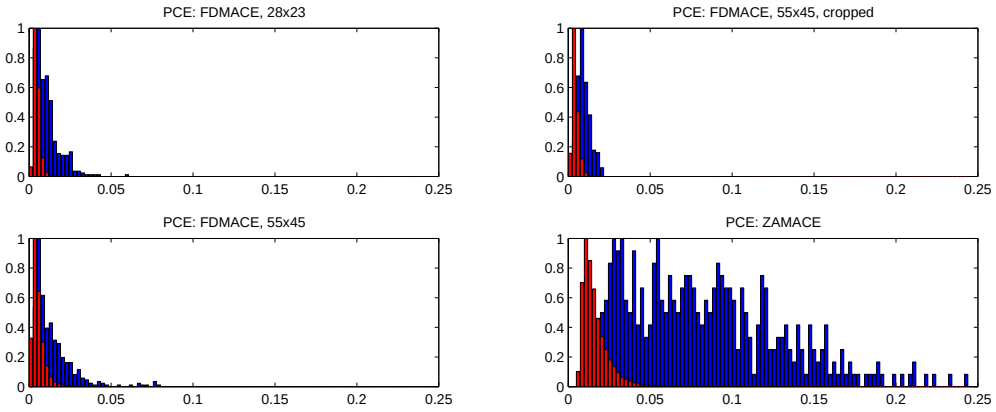
With the ZACF formulations we have currently presented, efficiently training ZACFs for training images much bigger than 28×23 becomes quite difficult. Therefore, it is very difficult to use the closed-form solutions presented here and test on datasets containing full resolution



(a) Distributions of Maximum Peak Scores



(b) Distributions of Output Correlation Energy



(c) Distributions of PCE

Figure 3.12: Distributions showing the peak scores, output correlation energy, and PCE for authentic (blue) and imposter (red) test images.

Table 3.2: EER for Different CF Formulations

Filter Name	Parameters	Original Formulation EER	Zero-Aliasing Formulation EER
MACE	n/a	0.170	**0.0856**
OTSDF	$\lambda = 0.5$	0.148	**0.108**
	$\lambda = 0.7$	0.136	**0.0925**
	$\lambda = 0.9$	0.132	**0.0800**
MVSDF	n/a	**0.143**	**0.143**
MSESDF	$\sigma^2 = 0$	0.170	**0.0856**
	$\sigma^2 = 1$	0.183	**0.0878**
	$\sigma^2 = 2$	0.199	**0.103**
	$\sigma^2 = 5$	0.255	**0.153**
UOTSDF	$\lambda = 0.5$	0.138	**0.106**
	$\lambda = 0.7$	0.137	**0.100**
	$\lambda = 0.9$	0.128	**0.0950**
MACH	$\lambda = 1, \psi = 0.5$	0.170	**0.0984**
UMACE	n/a	0.168	**0.103**
MMCF	$\lambda = 0.5$	0.138	**0.106**
	$\lambda = 0.7$	0.136	**0.0999**
	$\lambda = 0.9$	0.128	**0.0950**
ASEF	$\sigma^2 = 0$	0.180	**0.105**
	$\sigma^2 = 1$	0.135	**0.0900**
	$\sigma^2 = 2$	0.170	**0.100**
	$\sigma^2 = 5$	0.268	**0.140**
MOSSE	$\sigma^2 = 0$	0.168	**0.103**
	$\sigma^2 = 1$	0.105	**0.0745**
	$\sigma^2 = 2$	0.140	**0.090**
	$\sigma^2 = 5$	0.253	**0.137**

images. In the next chapter, we will explore several alternative formulations that better allow us to train ZACFs on larger training images. We then present results on larger datasets.

3.9 Conclusions and Discussion

In this chapter, we have introduced our zero-aliasing solution to the aliasing issue in CFs. We first used the MACE filter to illustrate the concept, and then we extended the idea to other CF designs. In the next chapter we will discuss alternative solutions to the problem and will present experimental results. We summarize the contributions of this chapter below.

- The formulation problem with existing CFs is that the tail of the CF template is non-zero. We force this tail to be zero by introducing zero-aliasing constraints into the optimization step of CF design.
- We illustrate the concept of ZACFs with the MACE filter, comparing the old approach (FDMACE), the time/space domain approach (TDMACE), and our new approach (ZAMACE). We show that ZAMACE outperforms the conventional FDMACE approach in terms of yielding lower unaliased ACE on the training set.
- We show that the ZAMACE filter converges to the TDMACE filter when the zero padding is equal to $N_x - 1$. Therefore, the ZAMACE formulation acts as a bridge between the FDMACE formulation (no zero-padding) and the TDMACE formulation (full zero-padding).
- We extend the zero-aliasing approach to 2D.
- We extend zero-aliasing constraints to the OTSDF filter.
- We extend zero-aliasing constraints to the class of generalized CFs, UCF and CCF.
- We derive the ZAMACH filter and show that it yields the same answer at the ZAUCF filter with the appropriate parameters selected.
- We show how zero-aliasing constraints can be added to the ASEF and MOSSE filter formulations.

- We extend zero-aliasing constraints to the MMCF formulation.

While we have not extended zero-aliasing constraints to every type of CF, it should become obvious how to extend the constraints to other CFs not covered here. Note that the UCF and CCF formulations generalize a large number of existing CF designs, which is a useful tool for extending zero-aliasing constraints to CFs not mentioned here.

Chapter 4

Alternatives to Zero-Aliasing Correlation Filters

In the previous chapter, we introduced ZACFs. This formulation completely fixes the aliasing issue found in CFs. Despite having closed form expressions for ZACFs for multiple CF designs, computation of these ZACFs can still be extremely challenging, because the constraint matrix $\mathbf{A}^+$ is quite large ($N^\# - N_x^\# \times N^\#$). Thus forming this matrix is a memory challenge, and performing computations (especially matrix inversions) on the resulting large matrices is very computationally intensive. In this chapter, we present three alternate formulations to the problem that perform better computationally. They are reduced-aliasing correlation filters, tail energy minimization, and proximal gradient methods. These formulations are helpful to making ZACFs practical for real data. At the end of this chapter, we present results on larger datasets.

4.1 Reduced-Aliasing Correlation Filters

In this section, we introduce the Reduced-Aliasing Correlation Filters (RACFs). RACFs are very closely related to ZACFs and can be viewed as a direct extension. As mentioned previously, the ZACF closed form solution requires the formation and subsequent inversion of a very

large matrix. This matrix's size is a function of the number of zero-aliasing constraints (i.e., the number of elements in the tail that we constrain to zero) and the DFT size used. In this section, we explore different methods for reducing the number of these constraints, which in turn makes the matrices smaller, reducing memory and computational burden. Here, we will focus on the MACE formulation; however, this approach is relevant to other CF designs as well.

Recall the ZAMACE formulation:

$$\bar{\mathbf{h}} = \mathbf{D}^{-1}\mathbf{B}\left(\mathbf{B}^{+}\mathbf{D}^{-1}\mathbf{B}\right)^{-1}\mathbf{k} \tag{4.1}$$

In this formulation, we must form the matrix $\mathbf{B}^{+}\mathbf{D}^{-1}\mathbf{B}$ which, for a D-dimensional application, is of size $(Q + N^{\#} - N_x^{\#}) \times (Q + N^{\#} - N_x^{\#})$, where

$$N^{\#} = \prod_{d=1}^{D} N_d \tag{4.2}$$

and

$$N_x^{\#} = \prod_{d=1}^{D} N_{x,d} \tag{4.3}$$

Note that the matrix size is a function of the number of zero-aliasing constraints that we impose, $p = N^{\#} - N_x^{\#}$. Therefore, if we impose fewer constraints, we can make this matrix smaller, which will require less computations and less memory. There are several ways to do this, which we explore below.

4.1.1 Method 1: Reducing the DFT Size

The first method that we can employ to reduce the number of constraints is to simply reduce the DFT size ($N_{x,d} < N_d < 2N_{x,d} - 1$). Doing so means that there is still aliasing due to circular correlation; however, we have noticed that this aliasing is significantly reduced, leading to very good results while improving the computational requirements. As noted in the previous chapter

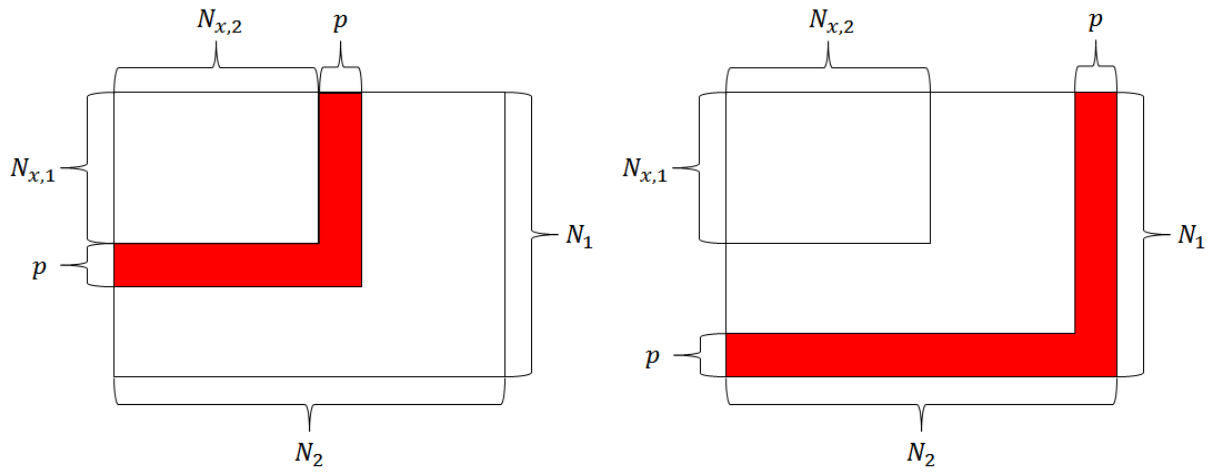
(see Fig. 3.3), we observed that the unaliased ACE for the zero-aliasing formulation decreases sharply as zero padding increased. The obtained filter for a small amount of zero padding exhibits an unaliased ACE that is almost as low as the unaliased ACE of the zero-aliasing filter obtained with full zero padding. Moreover, we notice that the PCE values on the training set for these instances are quite good, indicating very sharp peaks and implying that a partially padded zero-aliasing filter can perform well.

4.1.2 Methods 2-5: Full-Size DFT with Fewer Constraints

The second general method that we employ uses full sized DFTs, i.e. $N_d = 2N_{x,d} - 1$. We then allow for a customized constraint mask, such that we only constrain a portion of the correlation template. The motivation for doing this is to reduce the energy in the template tail. Recall the 2D formulation of the ZACF, which employed a selection matrix $\mathbf{S}$ defined in Eq. 3.17. In this case we change the selection matrix to a specialized pattern which constrains a portion of the template tail. We form matrix $\mathbf{S}$ by placing 1 at these constrained pixel locations and placing 0 elsewhere. We then determine $\mathbf{s}_v$ as in Eq. 3.18. Then, as before, we form form matrix $\mathbf{A}^+$ by selecting the rows of matrix $\mathbf{\Omega\Phi}$ corresponding to entries in vector $\mathbf{s}_v$ that are equal to 1. We experiment with different methods of doing this, and describe each below. We do this to gain insight as to which part of the tail is important to constrain.

4.1.2.1 Method 2: Full Tail

We illustrate the first two methods in Fig. 4.1 . In Method 2A, we set the spatial coordinates closest to the training images to zero. In Method 2B, we set the spatial coordinates at the edge of the template tail to zero. For Method 2A, we never use DFT sizes larger than $2N_{x,d} - 1$, so if a value of p for Method 2A requires a DFT size larger than $2N_{x,d} - 1$, the DFT size is simply clipped to size $2N_{x,d} - 1$. Similarly, for Method 2B, in no case do we constrain any of the pixels within the location of the training image, i.e. pixels 1 to $N_{x,d}$ are never constrained (note that



(a) Method 2A (b) Method 2B

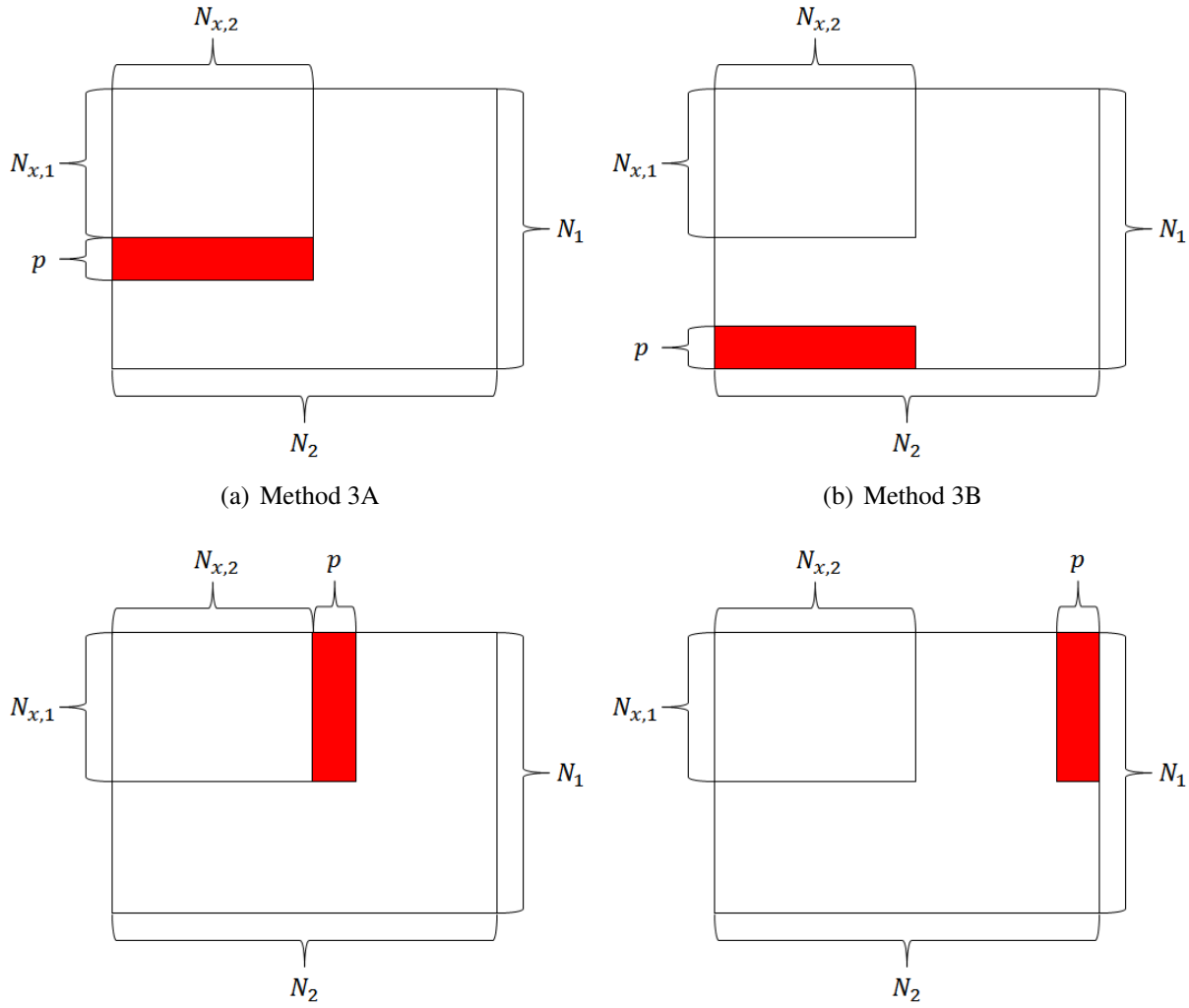
Figure 4.1: Method 2: Constraining the inner and outer portions of the tail.

image indexing starts at 1). If we use a value of $p > N_d - N_{x,d}$, the number of constraints in that dimension is simply clipped at $N_d - N_{x,d}$. We mention these details because they are relevant when $N_{x,1} \neq N_{x,2}$.

4.1.2.2 Method 3: Single Tail

Method 3 is very similar to Method 2. In this case, we only constrain the tail in a single dimension, i.e. we only constrain the spatial values in the columns (or rows) corresponding to the training image location. We illustrate the four different ways of doing this in Fig. 4.2.

As before, for Methods 3A and 3C, we never use DFT sizes larger than $2N_{x,d} - 1$, so if a value of p requires a DFT size larger than $2N_{x,d} - 1$, the DFT size is simply clipped to size $2N_{x,d} - 1$. Similarly, for Methods 3B and 3D, in no case do we constrain any of the pixels within the location of the training image as discussed previously (for values of $p > N_d - N_{x,d}$, the number of constraints is simply clipped to $N_d - N_{x,d}$).



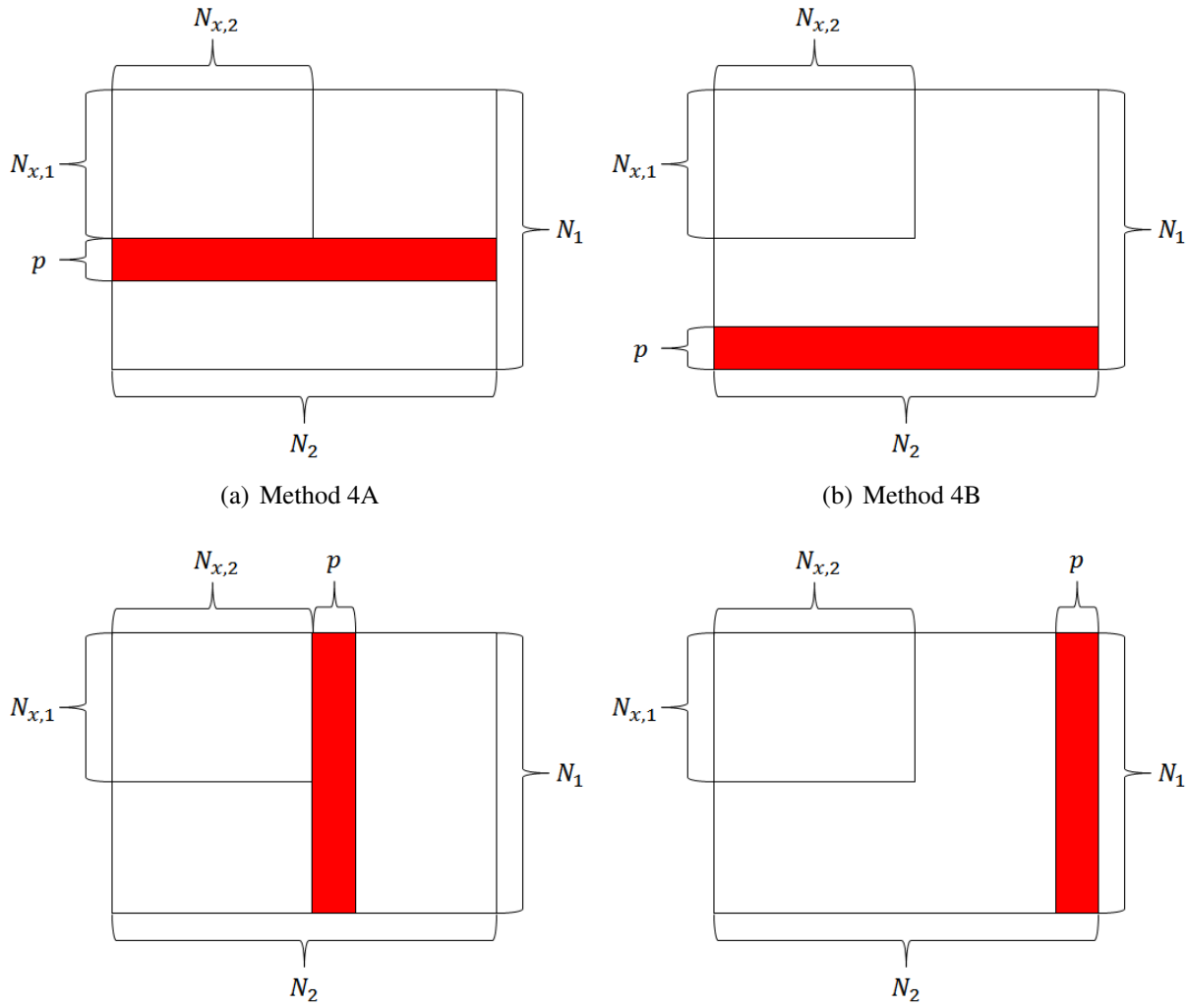
(a) Method 3A

(b) Method 3B

(c) Method 3C

(d) Method 3D

Figure 4.2: Method 3: Constraining a single tail.



(a) Method 4A

(b) Method 4B

(c) Method 4C

(d) Method 4D

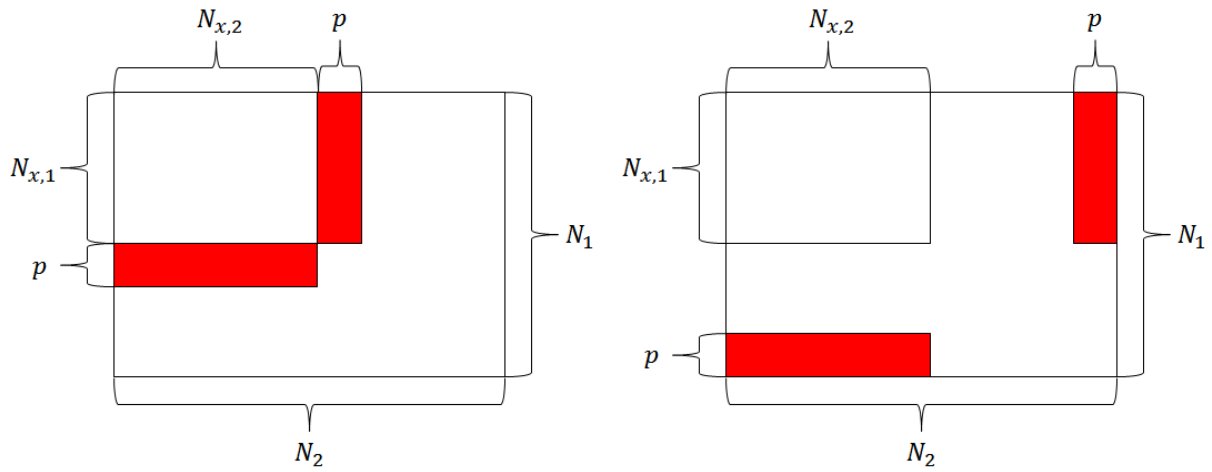
Figure 4.3: Method 4: Constraining a widened single tail.

4.1.2.3 Method 4: Widened Single Tail

Method 4 is the same as Method 3, except it uses a full sized (wider) tail. It is illustrated in Fig. 4.3.

4.1.2.4 Method 5: Double Tail

Method 5 essentially combines the constraints in Method 3 but enforces them for the tails in both dimensions. It is illustrated in Fig. 4.4.



(a) Method 5A (b) Method 5B

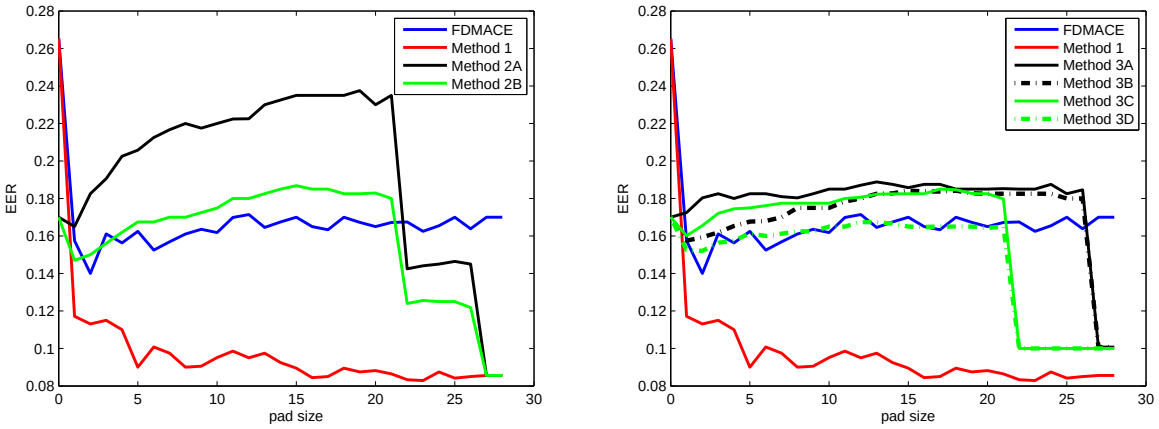
Figure 4.4: Method 5: Constraining a double tail.

4.1.3 Experimental Results

In this section, we compare a standard FDMACE formulation to the reduced-aliasing implementations described in Methods 1-5. In the case of the FDMACE formulation, we use a DFT size of $\min(N_{x,d} + p, 2N_{x,d} - 1)$. We again test on the AT&T/ORL dataset [43] as described in the previous chapter, using downsampled images of size 28×23 and 10-fold cross validation. We sweep parameter p and show our results in terms of EER.

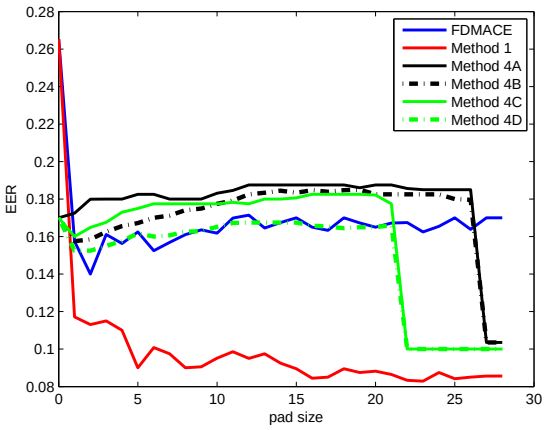
We compare FDMACE to Methods 1, 2A, and 2B in Fig. 4.5a. Note that Method 1 outperforms the other methods by a large margin. In addition, the EER is quite low for $p \geq 5$, confirming our previous observations of the trained filter, which exhibited a low ACE and a high PCE on the train set. Method 2A and 2B actually perform worse than the FDMACE formulation until $p \geq 22$, which is $N_{d,2} - N_{x,2}$ for these images. This indicates that completely constraining the tail in one dimension leads to a dramatic increase in performance. We notice a similar drop off for $p \geq 27$, which is $N_{d,1} - N_{x,1}$. Note that at $p = 27$, Method 1, 2A, and 2B converge to the standard ZAMACE. For $p = 0$, Method 1 and FDMACE are the same. Finally, the FDMACE at $p = 27$ is the same as the Method 2A and 2B when there are no constraints ($p = 0$).

We compare Method 1 to Method 3 in Fig. 4.5b. Note that in this case, we are only con-

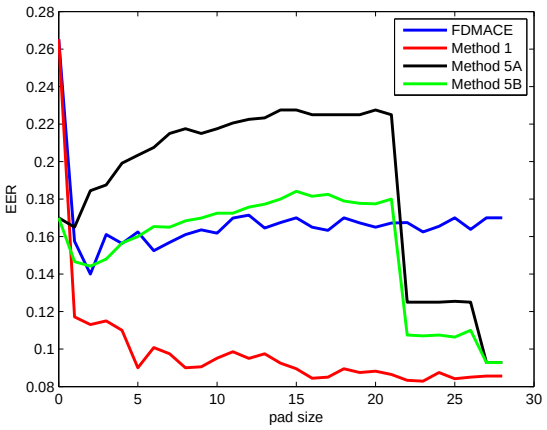


(a)

(b)



(c)



(d)

Figure 4.5: EER as a function of parameter p.

straining spatial values of a single tail. Note that for a small number of constraints (small p), the performance is not any better than the FDMACE. However, once $p = N_d - N_{x,d}$ (i.e. we fully constrain the entire tail in one dimension), the performance becomes drastically better. This is shown by the EER drops at $p = 22$ and $p = 27$ for Methods 3C/D and 3A/B, respectively. Note that Method 3B outperforms 3A, and likewise 3D outperforms 3C. This indicates that constraining the tip of the tail (the portion farthest from the image location) is more important than constraining the base of the tail (the portion closest to the image location).

We show Method 4 in Fig. 4.5c. Here, we notice the same trends that were observed for Method 3. Note that the performance of Method 4 is approximately the same as that of Method 3. This indicates that the added constraints (wider tail) as found in Method 4 does not provide any benefit.

Finally, we show Method 5 in Fig. 4.5d. Note that the results here largely resemble our observations for Method 2. Note, however, that Method 5A outperforms Method 2A, and Method 5B outperforms Method 2B. This indicates that the constraints located in the corner of the constraint regions for Method 2A and 2B do little to improve performance.

We summarize our observations for Methods 2-5 here:

- When constraining a single template tail, the entire tail should be constrained;
- Constraining the tip of a tail leads to better results than constraining the base of the tail;
- Using a wider tail (Method 3) leads to very little improvement;
- Adding constraints in the corner of the constraint region (opposite the image location) does not tend to improve performance (compare Method 2 and 5).

This evaluation leads us to the conclusion that, in practice, Method 1 is the most promising method. We completed a thorough evaluation of Methods 2-5 to demonstrate that, even though there are many different ways to constrain a portion of the correlation template, these methods all fall short of Method 1 in terms of recognition performance.

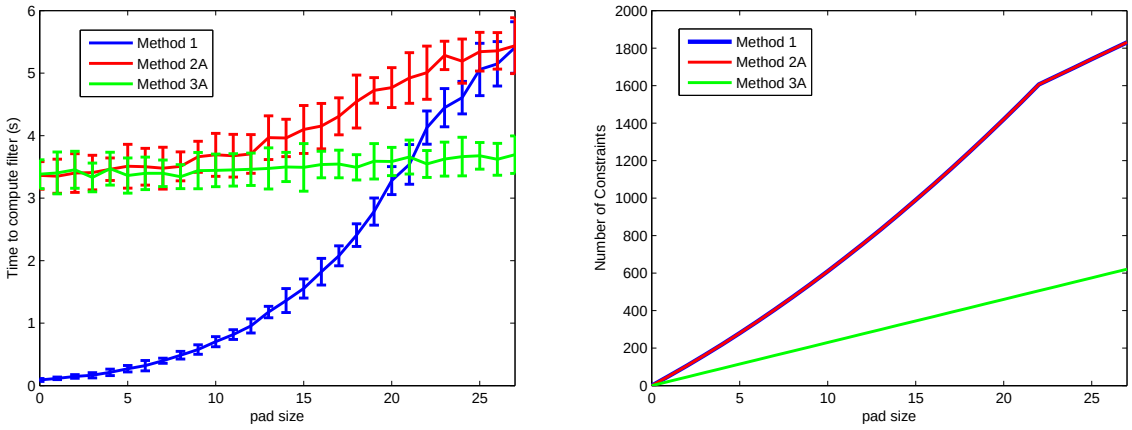
4.1.4 Computational Considerations

We now perform a computational evaluation on the time required to compute these CFs. For simplicity, we only consider Methods 1, 2A, and 3A. For each experiment, we train the filter with 9 training images of size 28×23. We then vary the pad size, p, and time how long it takes to compute each CF. We repeat this 20 times for each pad size and report the average results. The platform for our experiment was a desktop running MATLAB 2011a with Windows 7, an Intel Core i7-2600 CPU (3.4 GHz), and 16 GB of RAM.

The results of this evaluation are shown in Fig. 4.6a. In this experiment, we used a maximum DFT size of 55×45 for Method 1 (i.e., we used $N_d = \min(N_{x,d} + p, 2N_{x,d} - 1)$ as before). The DFT size for Methods 2A and 3A are always 55×45. We also show the number of constraints in Fig. 4.6b. Note that Method 1 and Method 2A have the same number of constraints for a given p. These results illustrate that the Method 1 CF can be computed considerably faster than both the other methods and the standard ZACF (equivalent to Method 1 for $p \geq 27$). For example, the ZACF took an average time of 5.41 seconds to compute, whereas the CF using Method 1 and $p = 5$ took 0.27 seconds (~ 20 times faster).

In addition to this, Method 1 is more memory efficient, as it uses smaller DFTs. Because Method 1 performs better at smaller value of p than other methods, Method 1 effectively requires fewer constraints than other methods, thus reducing the memory constraints of computing the $\mathbf{A}$ matrix. For example, in Fig. 4.6b, Method 3A has fewer constraints than Method 1 for a given p. However, Method 1 at $p = 5$ can achieve better recognition than Method 3A at $p \geq 27$. At these points, Method 1 requires fewer constraints (and therefore less memory) than Method 3A.

We have shown that Method 1 exhibits both the best recognition and computational performance. Therefore, Method 1 is the recommended method. From this point forward, we will refer to Method 1 simply as the RACF.



(a) Time to compute CF

(b) Number of Constraints

Figure 4.6: Comparison of the computational complexity of computing the ZACF using Methods 1, 2A, and 3A. The error bars in (a) indicate the 95% confidence interval. Note that the blue curve in (b) is hidden behind the red curve.

4.2 Tail Energy Minimization

In the previous section, we introduced RACFs, which focused on reducing the number of zero-aliasing constraints necessary to compute ZACFs. This in turn reduced the computational and memory requirements for computing CFs. In this section, we present an alternate formulation of ZACFs. This formulation completely removes the zero-aliasing constraints and replaces them with a tail energy minimization term. This results in sparse matrices that allow for faster computation of CFs. This approach is also beneficial in that it does not require significant reformulation of the closed-form expressions of existing CF designs. In this section, we use the MACE filter as an example, but it will become apparent that this method is trivially extended to other CF designs.

4.2.1 1D Formulation

We first present the formulation in 1D and later extend it to 2D. We refer to this method as Tail Energy Minimization (TEM) because we replace the hard constraints that force the tail of the

filter to zero with a term that represents the total energy in the template tail.

Let us assume we have Q training signals (1D) of length N_x samples. We wish to build a zero-aliasing correlation filter by using a DFT size of $N = 2N_x - 1$. For the ZACF method we would introduce $N - N_x$ constraints. However, in this case, we want to minimize the energy in the tail of the *spatial domain* template,

$$\min_{\mathbf{h}} \mathbf{h}^{+}\mathbf{S}\mathbf{h} \tag{4.4}$$

In this case, the matrix $\mathbf{S}$ is a diagonal matrix of size $N \times N$. The main diagonal is formed from the vector $[\ \mathbf{0}_{1\times N_x}\ \ \mathbf{1}_{1\times(N-N_x)}\ ]$. This matrix serves to "select" the tail portion of the filter that is to be minimized. Recall that $\mathbf{h} = \mathbf{F}_N^{-1}\bar{\mathbf{h}}$, where $\mathbf{F}_N^{-1}$ is the $N \times N$ IDFT matrix. Substituting this into Eq. 4.4, we obtain

$$\begin{aligned} &\min_{\bar{\mathbf{h}}} (\mathbf{F}_N^{-1}\bar{\mathbf{h}})^{+}\mathbf{S}\mathbf{F}_N^{-1}\bar{\mathbf{h}} \\ &= \min_{\bar{\mathbf{h}}} \bar{\mathbf{h}}^{+}(\mathbf{F}_N^{-1})^{+}\mathbf{S}\mathbf{F}_N^{-1}\bar{\mathbf{h}} \\ &= \min_{\bar{\mathbf{h}}} \bar{\mathbf{h}}^{+}\bar{\mathbf{S}}\bar{\mathbf{h}} \end{aligned} \tag{4.5}$$

where

$$\bar{\mathbf{S}} = (\mathbf{F}_N^{-1})^{+}\mathbf{S}\mathbf{F}_N^{-1} \tag{4.6}$$

To derive the tail energy minimized MACE (TEMMACE) filter, we now minimize the function

$$\min_{\bar{\mathbf{h}}} \bar{\mathbf{h}}^{+}\mathbf{D}\bar{\mathbf{h}} + \lambda_{TEM}\bar{\mathbf{h}}^{+}\bar{\mathbf{S}}\bar{\mathbf{h}}$$
$$= \min_{\bar{\mathbf{h}}} \bar{\mathbf{h}}^{+}\mathbf{T}\bar{\mathbf{h}} \tag{4.7}$$

subject to the constraints (originally presented in Eq. 2.8)

$$\bar{\mathbf{X}}^{+}\bar{\mathbf{h}} = N\mathbf{u} \tag{4.8}$$

Here, λ_{TEM} is a free parameter that controls the tradeoff between minimizing ACE and tail energy (TE), and

$$\mathbf{T} = \mathbf{D} + \lambda_{TEM}\bar{\mathbf{S}} \tag{4.9}$$

Because this is in the same form as the original MACE formulation, the TEMMACE is given by

$$\bar{\mathbf{h}} = \mathbf{T}^{-1}\bar{\mathbf{X}}\left(\bar{\mathbf{X}}^{+}\mathbf{T}^{-1}\bar{\mathbf{X}}\right)^{-1}\mathbf{u} \tag{4.10}$$

4.2.2 2D Formulation

We now adapt this formulation to higher dimensional signals. We demonstrate this for 2D, but note that similar manipulations can be done for higher dimensional signals.

In 2D, we want to minimize the tail energy for every row *and* every column in the resulting filter. To do this, we note the linearity of the DFT. Refer to Fig. 4.7. Observe that the 2D DFT ($\bar{\mathbf{h}}$) of the spatial domain template $\mathbf{h}$ may be computed in two ways: first, by taking the DFT of the rows and then by taking the DFT of the columns; and second, by taking the DFT of the columns and then taking the DFT of the rows. Note that the template in the spatial domain has zeros in the locations $y > N_{x,1}$ and $x > N_{x,2}$. Note that if we took the DFT first along the second dimension

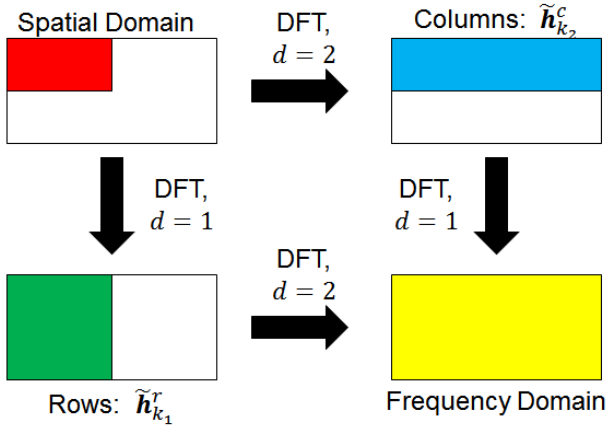


Figure 4.7: Demonstration of the linearity of the DFT operation in 2D.

(DFT of each row), the resulting array has zeros in locations $y > N_{x,1}$. On the other hand, if we take the DFT first along the first dimension (DFT of each column), the resulting array has zeros in the locations $x > N_{x,2}$. In both cases, taking the DFT along the opposite direction as the first step results in the full 2D DFT. We take advantage of these manipulations to separately minimize the row tail energy (RTE) and column tail energy (CTE). Starting with the 2D DFT, we first take the IDFT along dimension 1 and then minimize the CTE of vectors $\tilde{\mathbf{h}}^c_{k_2}$. Second, (again, starting with the 2D DFT), we take the IDFT along dimension 2 and then minimize the RTE of vectors $\tilde{\mathbf{h}}^r_{k_1}$.

We formulate the 2D TEMMACE as follows. First, the tail energy is given by

$$TE = \sum_{k_1=1}^{N_1} \tilde{\mathbf{h}}^{r+}_{k_1} \mathbf{S}_r \tilde{\mathbf{h}}^r_{k_1} + \sum_{k_2=1}^{N_2} \tilde{\mathbf{h}}^{c+}_{k_2} \mathbf{S}_c \tilde{\mathbf{h}}^c_{k_2} \tag{4.11}$$

Here, $\tilde{\mathbf{h}}^r_{k_1}$ and $\tilde{\mathbf{h}}^c_{k_2}$ are the rows and the columns respectively of the two intermediate-domain arrays as shown in Fig. 4.7. The matrices $\mathbf{S}_r$ and $\mathbf{S}_c$ are diagonal matrices given by

$$\mathbf{S}_r = diag([\ \mathbf{0}_{1\times N_{x,2}} \quad \mathbf{1}_{1\times(N_2-N_{x,2})}\ ]^T) \tag{4.12}$$

$$\mathbf{S}_c = diag([\ \mathbf{0}_{1\times N_{x,1}} \quad \mathbf{1}_{1\times(N_1-N_{x,1})}\ ]^T) \tag{4.13}$$

Our goal is to express $\tilde{\mathbf{h}}^r_{k_1}$ and $\tilde{\mathbf{h}}^c_{k_2}$ in terms of $\bar{\mathbf{h}}$. Recall that $\bar{\mathbf{h}}$ is a vectorized 2D DFT[1] of the correlation filter $H(k,l)$. We start with the case of the columns of the 2D DFT $H(k,l)$, which we denote as $\bar{\mathbf{h}}^c_{k_2}$. We use a selection matrix to select these columns from the vector $\bar{\mathbf{h}}$. This is given by

$$\bar{\mathbf{h}}^c_{k_2} = \mathbf{V}^c_{k_2}\bar{\mathbf{h}} \tag{4.14}$$

where the matrix $\mathbf{V}^c_{k_2}$ is given by

$$\mathbf{V}^c_{k_2} = \left[\ \mathbf{0}_{N_1\times(N_1(k_2-1))} \quad \mathbf{I}_{N_1} \quad \mathbf{0}_{N_1\times(N_1N_2-N_1k_2)}\ \right] \tag{4.15}$$

Note that the vector $\bar{\mathbf{h}}^c_{k_2}$ is $N_1 \times 1$. Next, we must take the IDFT of each of these vectors to obtain $\tilde{\mathbf{h}}^c_{k_2}$,

$$\begin{aligned}\tilde{\mathbf{h}}^c_{k_2} &= \mathbf{F}^{-1}_{N_1}\bar{\mathbf{h}}^c_{k_2} \\ &= \mathbf{F}^{-1}_{N_1}\mathbf{V}^c_{k_2}\bar{\mathbf{h}}\end{aligned} \tag{4.16}$$

Similarly, we denote the $N_2 \times 1$ vector formed from row k_1 of the 2D DFT $H(k,l)$ as $\bar{\mathbf{h}}^r_{k_1}$. We use a selection matrix to select these rows from the vector $\bar{\mathbf{h}}$. This is given by

$$\bar{\mathbf{h}}^r_{k_1} = \mathbf{V}^r_{k_1}\bar{\mathbf{h}} \tag{4.17}$$

[1]We assume column-by-column vectorization (i.e. each column is concatenated).

where the matrix $\mathbf{V}_{k_1}^r$ is given by

$$\mathbf{V}_{k_1}^r = circshift(\mathbf{V}^r, k_1 - 1) \tag{4.18}$$

where the $circshift()$ operator circularly shifts the matrix $\mathbf{V}^r$ to the right by the specified number of entries. The matrix $\mathbf{V}^r$ is given by

$$\mathbf{V}^r = \begin{bmatrix} \boldsymbol{\delta}_1 & \mathbf{0}_{N_2 \times N_1 - 1} & \boldsymbol{\delta}_2 & \mathbf{0}_{N_2 \times N_1 - 1} & \cdots & \boldsymbol{\delta}_{N_2} & \mathbf{0}_{N_2 \times N_1 - 1} \end{bmatrix} \tag{4.19}$$

where the vector $\boldsymbol{\delta}_c$ is an all zero $N_2 \times 1$ vector with a "one" in the cth row. Next, we must take the IDFT of each of these vectors to obtain $\tilde{\mathbf{h}}_{k_1}^r$,

$$\begin{aligned} \tilde{\mathbf{h}}_{k_1}^r &= \mathbf{F}_{N_2}^{-1} \bar{\mathbf{h}}_{k_1}^r \\ &= \mathbf{F}_{N_2}^{-1} \mathbf{V}_{k_1}^r \bar{\mathbf{h}} \end{aligned} \tag{4.20}$$

Next we substitute these expressions for $\tilde{\mathbf{h}}_{k_1}^r$ and $\tilde{\mathbf{h}}_{k_2}^c$ into Eq. 4.11 to obtain

$$\begin{aligned} TE &= \sum_{k_1=1}^{N_1} \bar{\mathbf{h}}^+ \mathbf{V}_{k_1}^{r+} (\mathbf{F}_{N_2}^{-1})^+ \mathbf{S}_r \mathbf{F}_{N_2}^{-1} \mathbf{V}_{k_1}^r \bar{\mathbf{h}} + \sum_{k_2=1}^{N_2} \bar{\mathbf{h}}^+ \mathbf{V}_{k_2}^{c+} (\mathbf{F}_{N_1}^{-1})^+ \mathbf{S}_c \mathbf{F}_{N_1}^{-1} \mathbf{V}_{k_2}^c \bar{\mathbf{h}} \\ &= \sum_{k_1=1}^{N_1} \bar{\mathbf{h}}^+ \mathbf{R}_{k_1}^r \bar{\mathbf{h}} + \sum_{k_2=1}^{N_2} \bar{\mathbf{h}}^+ \mathbf{R}_{k_2}^c \bar{\mathbf{h}} \\ &= \bar{\mathbf{h}}^+ \left(\sum_{k_1=1}^{N_1} \mathbf{R}_{k_1}^r + \sum_{k_2=1}^{N_2} \mathbf{R}_{k_2}^c \right) \bar{\mathbf{h}} \\ &\bar{\mathbf{h}}^+ \mathbf{R} \bar{\mathbf{h}} \end{aligned} \tag{4.21}$$

where

$$\mathbf{R}_{k_1}^r = \mathbf{V}_{k_1}^{r+}(\mathbf{F}_{N_2}^{-1})^+\mathbf{S}_r\mathbf{F}_{N_2}^{-1}\mathbf{V}_{k_1}^r \tag{4.22}$$

$$\mathbf{R}_{k_2}^c = \mathbf{V}_{k_2}^{c+}(\mathbf{F}_{N_1}^{-1})^+\mathbf{S}_c\mathbf{F}_{N_1}^{-1}\mathbf{V}_{k_2}^c \tag{4.23}$$

$$\mathbf{R} = \sum_{k_1=1}^{N_1}\mathbf{R}_{k_1}^r + \sum_{k_2=1}^{N_2}\mathbf{R}_{k_2}^c \tag{4.24}$$

For the TEMMACE, we minimize the function

$$\begin{aligned}\min_{\bar{\mathbf{h}}}\bar{\mathbf{h}}^+\mathbf{D}\bar{\mathbf{h}} + \lambda_{TEM}\bar{\mathbf{h}}^+\mathbf{R}\bar{\mathbf{h}}\\ = \min_{\bar{\mathbf{h}}}\bar{\mathbf{h}}^+\mathbf{T}\bar{\mathbf{h}}\end{aligned} \tag{4.25}$$

subject to the constraints (as in the original MACE formulation)

$$\bar{\mathbf{X}}^+\bar{\mathbf{h}} = N_1N_2\mathbf{u} \tag{4.26}$$

Here, λ_{TEM} is a free parameter that controls the tradeoff between minimizing ACE and TE, and

$$\mathbf{T} = \mathbf{D} + \lambda_{TEM}\mathbf{R} \tag{4.27}$$

The TEMMACE is given by

$$\bar{\mathbf{h}} = \mathbf{T}^{-1}\bar{\mathbf{X}}\left(\bar{\mathbf{X}}^+\mathbf{T}^{-1}\bar{\mathbf{X}}\right)^{-1}\mathbf{u} \tag{4.28}$$

In this section we derived the TEMMACE filter. It is easy to extend this to other filter types, which is a major advantage of the TEM formulation. Because most other filters minimize a quadratic term (ACE, ONV, MSE, etc.), the TEM term is simply combined with this term as illustrated in the MACE example.

4.2.3 Implementation Notes

Note that the matrices $\mathbf{S}_r$, $\mathbf{S}_c$, and subsequently the matrices $\mathbf{R}^c_{k_2}$ and $\mathbf{R}^r_{k_1}$ are sparse. This means that matrix $\mathbf{R}$, while not diagonal, is also sparse. Therefore, we may take advantage in using sparse matrix programming and storage techniques in implementation. Compared to the original ZACF formulation, the TEM formulation is faster and more memory efficient, which is important for implementation.

Note that $\mathbf{R}$ may be written as

$$\mathbf{R} = \mathbf{R}^r + \mathbf{R}^c \tag{4.29}$$

where $\mathbf{R}^r = \sum_{k_1=1}^{N_1} \mathbf{R}^r_{k_1}$ and $\mathbf{R}^c = \sum_{k_2=1}^{N_2} \mathbf{R}^c_{k_2}$. First, we examine matrix $\mathbf{R}^c$, which may be written as

$$\begin{aligned}
\mathbf{R}^c &= \sum_{k_2=1}^{N_2} \mathbf{R}^c_{k_2} \\
&= \sum_{k_2=1}^{N_2} \mathbf{V}^{c+}_{k_2} (\mathbf{F}^{-1}_{N_1})^{+} \mathbf{S}_c \mathbf{F}^{-1}_{N_1} \mathbf{V}^c_{k_2} \\
&= \sum_{k_2=1}^{N_2} \mathbf{V}^{c+}_{k_2} \mathbf{K}_1^{+} \mathbf{K}_1 \mathbf{V}^c_{k_2} \\
&= \sum_{k_2=1}^{N_2} \mathbf{V}^{c+}_{k_2} \mathbf{Z}_1 \mathbf{V}^c_{k_2}
\end{aligned} \tag{4.30}$$

where $\mathbf{K}_1$ represents the last $\mathbf{N}_1 - \mathbf{N}_{x,1}$ rows of the matrix $\mathbf{F}^{-1}_{N_1}$. Note that the matrix $\mathbf{Z}_1 = \mathbf{K}_1^{+}\mathbf{K}_1$ is a $N_1 \times N_1$ conjugate symmetric and Toeplitz matrix. The operation in Eq. 4.30 replicates matrix $\mathbf{Z}_1$, forming the block diagonal matrix, $\mathbf{R}^c$, which can be written as

$$\mathbf{R}^c = \begin{bmatrix} \mathbf{Z}_1 & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{Z}_1 & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{Z}_1 \end{bmatrix}$$
$$= \mathbf{I}_{N_2} \otimes \mathbf{Z}_1 \tag{4.31}$$

where $\mathbf{I}_{N_2}$ denotes an identity matrix of size N_2 and $\otimes$ denotes the Kronecker product. Similarly, we can write matrix $\mathbf{R}^r$ as

$$\begin{aligned} \mathbf{R}^r &= \sum_{k_1=1}^{N_1} \mathbf{R}^r_{k_1} \\ &= \sum_{k_1=1}^{N_1} \mathbf{V}^{r+}_{k_1} (\mathbf{F}^{-1}_{N_2})^{+} \mathbf{S}_r \mathbf{F}^{-1}_{N_2} \mathbf{V}^r_{k_1} \\ &= \sum_{k_1=1}^{N_1} \mathbf{V}^{r+}_{k_1} \mathbf{K}_2^{+} \mathbf{K}_2 \mathbf{V}^r_{k_1} \\ &= \sum_{k_1=1}^{N_1} \mathbf{V}^{r+}_{k_1} \mathbf{Z}_2 \mathbf{V}^r_{k_1} \end{aligned} \tag{4.32}$$

where $\mathbf{K}_2$ represents the last $\mathbf{N}_2 - \mathbf{N}_{x,2}$ rows of the matrix $\mathbf{F}^{-1}_{N_2}$. Note that the matrix $\mathbf{Z}_2 = \mathbf{K}_2^{+}\mathbf{K}_2$ is a $N_2 \times N_2$ conjugate symmetric and Toeplitz matrix. The operation in Eq. 4.32 is a 2D upsampling operation, in which zeros are placed between each entry of the matrix. However, since the matrix $\mathbf{R}^r$ is the sum of N_1 upsampling operations, the end result is a multi-diagonal matrix that is both Toeplitz and conjugate symmetric. We may write matrix $\mathbf{R}^r$ as

$$\mathbf{R}^r = \begin{bmatrix} Z_2(1,1) & 0 & \cdots & 0 & Z_2(1,2) & 0 & \cdots & 0 \\ 0 & Z_2(1,1) & 0 & \vdots & 0 & Z_2(1,2) & 0 & \vdots \\ \vdots & 0 & Z_2(1,1) & 0 & \vdots & 0 & \ddots & 0 \\ 0 & \vdots & 0 & Z_2(1,1) & 0 & \vdots & 0 & Z_2(1,2) \\ Z_2(2,1) & 0 & \vdots & 0 & Z_2(1,1) & 0 & \vdots & 0 \\ 0 & Z_2(2,1) & 0 & \vdots & 0 & Z_2(1,1) & \ddots & \vdots \\ \vdots & 0 & \ddots & 0 & \vdots & 0 & \ddots & 0 \\ 0 & \cdots & 0 & Z_2(2,1) & 0 & \cdots & 0 & Z_2(1,1) \end{bmatrix}$$

$$= \mathbf{Z}_2 \otimes \mathbf{I}_{N_1} \tag{4.33}$$

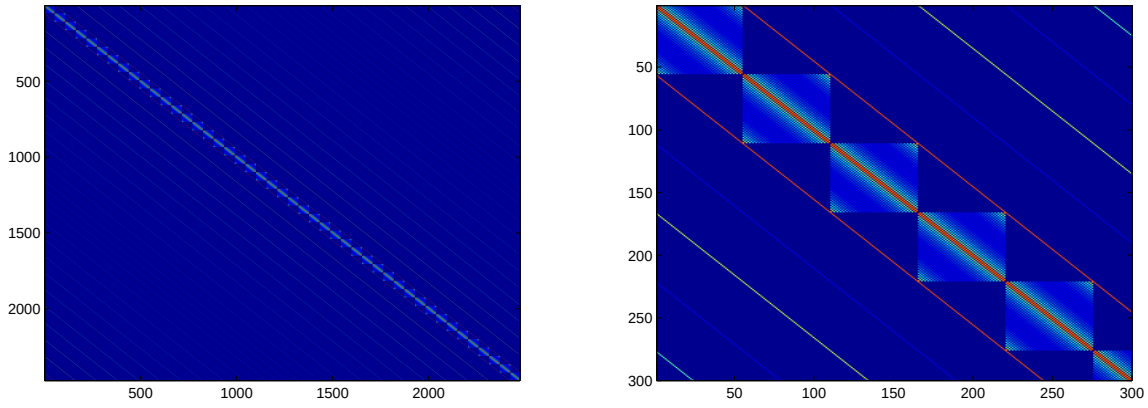
where $\mathbf{I}_{N_1}$ denotes an identity matrix of size N_1. Note that Eq. 4.31 and 4.33 may be used to quickly generate $\mathbf{R}^c$ and $\mathbf{R}^r$, respectively.

We form matrix $\mathbf{R}$ as the sum of matrices $\mathbf{R}^r$ and $\mathbf{R}^c$. Note that this matrix is not Toeplitz, but is conjugate symmetric. (The matrix is not Toeplitz because matrix $\mathbf{R}^c$ is not Toeplitz.) We show an example visualization of this matrix in Fig. 4.8. Note that matrix $\mathbf{T}$ has the same format as $\mathbf{R}$, as it effectively adds values along the diagonal. We have been unable to locate a good solution for inverting this matrix (or solving $\mathbf{y} = \mathbf{T}^{-1}\mathbf{x}$, where $\mathbf{y}$ and $\mathbf{x}$ are vectors) quickly by taking advantage of its structure.

4.2.4 Computational Considerations

In this section, we present a few brief experiments that demonstrate the superiority of TEM CFs over ZACFs in terms of computational speed. We present additional results in Sections 4.4 and 4.5.1.1.

To demonstrate the TEM vs. ZACF, we once again use the AT&T/ORL Face Dataset [43]. For each experiment, we train the filter with 9 training images of size 28×23. We repeat this 20



(a) Full Matrix

(b) Closeup, Main Diagonal

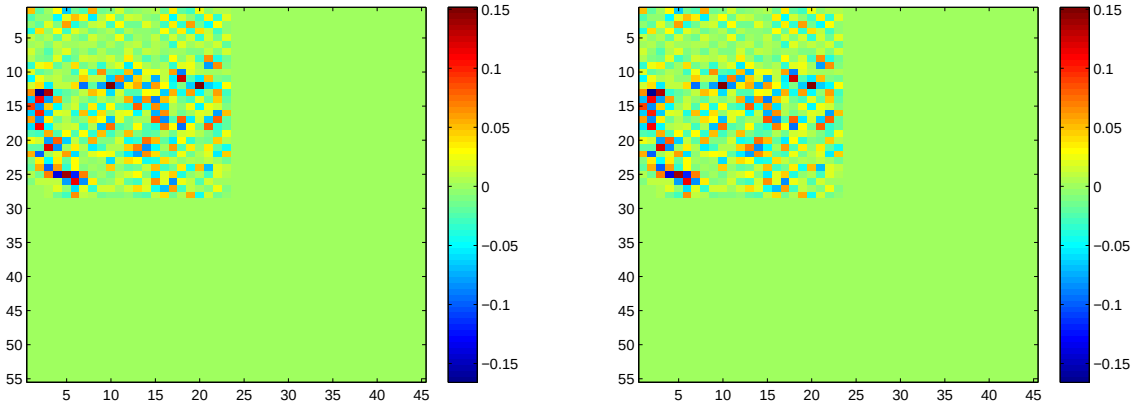
Figure 4.8: Visualization of matrix $\mathbf{R}$. In this example, $N_1 = 55$, $N_2 = 45$, which means $\mathbf{R}$ is $N_1N_2 \times N_1N_2$. Note that the blue color represents 0. This particular matrix is 96% sparse.

times and report the average results for both TEM and ZACF filters. We also present the MSE between the TEM and ZACF templates, after normalizing each to unit energy. For the TEM formulation, we use $\lambda_{TEM} = 100$. In general, this parameter affects MSE. We have observed that $\lambda_{TEM} = 100$ usually works quite well, however sometimes we get better performance for some filters with $\lambda_{TEM} = 10$. In general, selecting this parameter should be done on a case-by-case basis.

The platform for our experiment was a desktop running MATLAB 2011a with Windows 7, an Intel Core i7-2600 CPU (3.4 GHz), and 16 GB of RAM. These results are shown in Table 4.1 for several different filter types. Note that for ASEF, the timing difference is quite large; this is because 9 exact filters must be trained for each ASEF filter. To supplement these results, we show the ZAMACE, TEMMACE, and the error between the two in Fig. 4.9. Note that these plots, along with the MSE values in Table 4.1, indicate that the differences between the ZACF and TEM CFs are minimal.

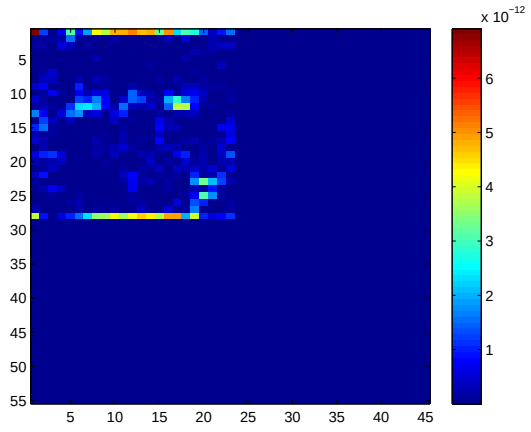
Table 4.1: Computational Comparison of of ZACF and TEM CFs

Filter	ZACF Time	TEM Time	MSE
MACE	2.26 s	0.38 s	1.14 e-13
OTSDF	2.37 s	0.39 s	2.92e-10
CCF	2.33 s	0.77 s	4.13e-10
UCF	2.31 s	0.38 s	8.08e-14
MMCF	5.87 s	0.75 s	1.34e-10
ASEF	50.54 s	3.89 s	8.46e-12
MOSSE	2.28 s	0.38 s	6.73e-12



(a) ZAMACE

(b) TEMMACE



(c) Squared Error

Figure 4.9: A comparison between ZAMACE and TEMMACE. The squared error plot shows the squared error between the two templates.

4.2.5 Reduced-Aliasing TEM

We may combine the idea behind RACFs with the TEM concept to derive so called reduced-aliasing TEM (RATEM) CFs. This idea is identical to that behind the RACF. A DFT of size $N_1 \times N_2$ is used to train the CF, with $N_d < 2N_{x,d} - 1$. Then, the energy in the tail of this CF is minimized as described previously. The actual formulation or RATEM is identical to that of TEM, except that the DFT size is different. We present additional results for RATEM in Sections 4.4 and 4.5.1.1.

4.3 Proximal Gradient Methods

In the previous sections, we illustrated alternatives to ZACF that yielded closed-form solutions. However, sometimes implementing these expressions may be impractical from a computational and memory perspective. An alternative solution is to use an iterative algorithm to solve for the ZACF. One iterative method to do this is using the proximal gradient descent method [48]. We have developed this method for both unconstrained and constrained CFs.

In general, a CF design involves a cost function $f(\bar{\mathbf{h}})$ with a corresponding gradient function $\nabla f(\bar{\mathbf{h}})$. Standard gradient descent [49] finds an optimal solution $\bar{\mathbf{h}}_{opt}$ by choosing an initial solution, $\bar{\mathbf{h}}_0$, and iteratively reduces the cost function $f(\bar{\mathbf{h}})$. Each iteration is computed as a function of the previous solution,

$$\bar{\mathbf{h}}_{k+1} = \bar{\mathbf{h}}_k - t_k \nabla f(\bar{\mathbf{h}}_k) \tag{4.34}$$

where t_k is the step size at iteration k. However, the standard gradient descent method does not allow for constraints to be imposed on the filter. To satisfy constraints, we apply the proximal gradient method [48], which is given by

$$\bar{\mathbf{h}}_{k+1} = prox\left(\bar{\mathbf{h}}_k - t_k \nabla f(\bar{\mathbf{h}}_k)\right) \tag{4.35}$$

Algorithm 4.1 Proximal Gradient for Unconstrained Filters

Compute $\bar{\mathbf{h}}_{conv}$ (conventional CF solution)
Initialize $\bar{\mathbf{h}}_0$ to $prox(\bar{\mathbf{h}}_{conv})$
repeat
  Compute $\bar{\mathbf{h}}_{k+1} = prox\left(\bar{\mathbf{h}}_k - t_k \nabla f(\bar{\mathbf{h}}_k)\right)$
until $\left|f(\bar{\mathbf{h}}_{k+1}) - f(\bar{\mathbf{h}}_k)\right| / \left|f(\bar{\mathbf{h}}_k)\right| < \epsilon$
Function $prox()$
1: Convert filter to 2D
2: Take 2D IDFT
3: Set template tail to zero
4: Take 2D DFT
5: Vectorize filter

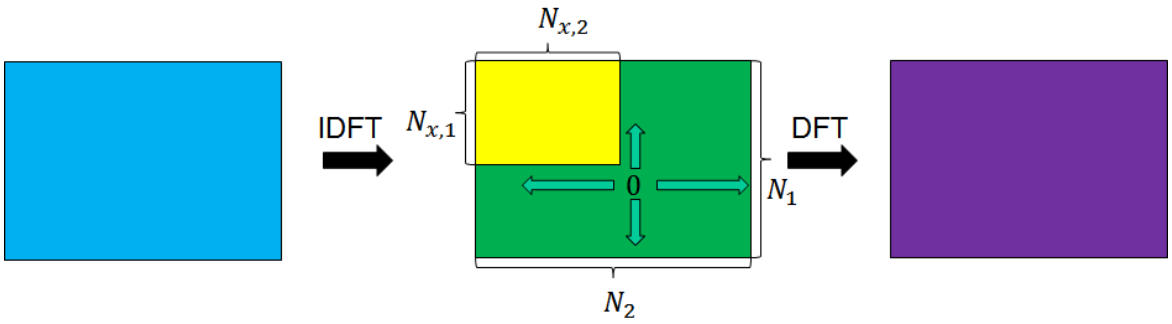
Here, the $prox()$ operator is an operation that imposes constraints on the filter. For unconstrained filters (e.g., ZAMOSSE), the $prox()$ operator (see Fig. 4.10a and Algorithm 4.1) transforms the filter update $\bar{\mathbf{h}}_k - t_k \nabla f(\bar{\mathbf{h}}_k)$ into the space domain and sets the template's tail to zero. It then transforms the resulting template back into the frequency domain to obtain $\bar{\mathbf{h}}_{k+1}$.

For the constrained case, the $prox()$ operator contains two steps (see Fig. 4.10b and Algorithm 4.2). First, the filter update $\bar{\mathbf{h}}_k - t_k \nabla f(\bar{\mathbf{h}}_k)$ is transformed into the space domain and the template's tail is set to zero (as in the unconstrained case). For the second step, we extract the main portion of the template and vectorize it. We denote this vector as $\mathbf{h}_k^{\#}$. Next, we form the vector $\mathbf{h}_k^{\#} + \mathbf{h}_\Delta$, convert it to 2D, substitute it back into the full template, and take the DFT to obtain $\bar{\mathbf{h}}_k$. We seek a vector $\mathbf{h}_\Delta$ such that

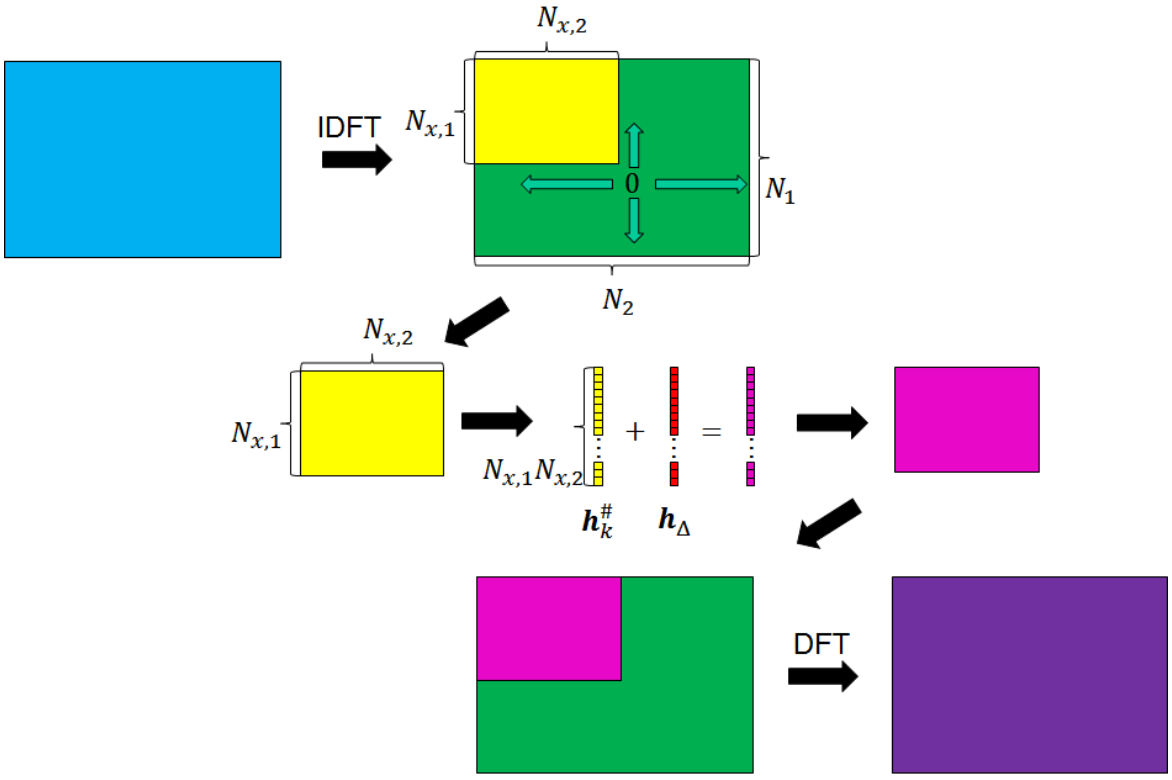
$$\mathbf{X}^T(\mathbf{h}_k^{\#} + \mathbf{h}_\Delta) = \mathbf{u} \tag{4.36}$$

Recall that $\mathbf{u}$ is a vector containing the desired peak constraints. Matrix $\mathbf{X}$ contains the vectorized padded training images in the space domain along each column. We want to pick a solution $\mathbf{h}_k^{\#} + \mathbf{h}_\Delta$ that satisfies the peak constraints, and that is closest to the current template $\mathbf{h}_k^{\#}$. Therefore, we minimize the L2 norm of $\mathbf{h}_\Delta$, i.e.

$$\min_{\mathbf{h}_\Delta} \mathbf{h}_\Delta^T \mathbf{h}_\Delta \tag{4.37}$$



(a) Unconstrained Case



(b) Constrained Case

Figure 4.10: Illustration of the proximal step, which is performed in the spatial domain. All DFTs and IDFTs are in 2D.

Algorithm 4.2 Proximal Gradient for Constrained Filters

Compute $\bar{\mathbf{h}}_{conv}$ (conventional CF solution)
Initialize $\bar{\mathbf{h}}_0$ to $prox(\bar{\mathbf{h}}_{conv})$
repeat
  Compute $\bar{\mathbf{h}}_{k+1} = prox\left(\bar{\mathbf{h}}_k - t_k \nabla f(\bar{\mathbf{h}}_k)\right)$
until $\left|f(\bar{\mathbf{h}}_{k+1}) - f(\bar{\mathbf{h}}_k)\right| / \left|f(\bar{\mathbf{h}}_k)\right| < \epsilon$
Function $prox()$

1: Convert filter to 2D
2: Take 2D IDFT
3: Set template tail to zero
4: Extract and vectorize $\mathbf{h}_k^{\#}$
5: Compute $\mathbf{h}_\Delta = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}(\mathbf{u} - \mathbf{X}^T\mathbf{h}_k^{\#})$
6: $\mathbf{h}_k^{\#} \leftarrow \mathbf{h}_k^{\#} + \mathbf{h}_\Delta$
7: Convert $\mathbf{h}_k^{\#}$ to 2D
8: Reinsert $\mathbf{h}_k^{\#}$ into template
9: Take 2D DFT
10: Vectorize filter

subject to $\mathbf{X}^T\mathbf{h}_\Delta = \mathbf{u} - \mathbf{X}^T\mathbf{h}_k^{\#}$. To accomplish this, we form the functional

$$\mathcal{L}(\mathbf{h}_\Delta, \boldsymbol{\omega}) = \mathbf{h}_\Delta^T\mathbf{h}_\Delta - 2\boldsymbol{\omega}^T(\mathbf{X}^T\mathbf{h}_\Delta - (\mathbf{u} - \mathbf{X}^T\mathbf{h}_k^{\#})) \tag{4.38}$$

Taking the gradient of $\mathcal{L}(\mathbf{h}_\Delta, \boldsymbol{\omega})$ with respect to $\mathbf{h}_\Delta$ and setting to $\mathbf{0}$, we obtain

$$\begin{aligned} 2\mathbf{h}_\Delta - 2\mathbf{X}\boldsymbol{\omega} &= \mathbf{0} \\ \mathbf{h}_\Delta &= \mathbf{X}\boldsymbol{\omega} \end{aligned} \tag{4.39}$$

Substituting this into Eq. 4.36, we obtain

$$\begin{aligned} \mathbf{X}^T(\mathbf{h}_k^{\#} + \mathbf{X}\boldsymbol{\omega}) &= \mathbf{u} \\ \boldsymbol{\omega} &= (\mathbf{X}^T\mathbf{X})^{-1}(\mathbf{u} - \mathbf{X}^T\mathbf{h}_k^{\#}) \end{aligned} \tag{4.40}$$

Finally, substituting this result back into Eq. 4.39 we obtain our solution as

$$\mathbf{h}_{\Delta} = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}(\mathbf{u} - \mathbf{X}^T\mathbf{h}_k^{\#}) \tag{4.41}$$

The proximal gradient solution ensures that the zero-aliasing constraints (and peak constraints, for constrained filters) are satisfied. This is done in the space domain, rather than forming matrix $\mathbf{A}$ (or matrix $\mathbf{B}$, for the constrained case) as in the closed form solution. This is advantageous because we save on the memory needed to compute and store $\mathbf{A}$ (or $\mathbf{B}$) and the subsequent computational resources needed to solve large systems of equations. The result is a memory-stable, low complexity solution that allows for efficient and fast computation of ZACFs.

4.4 Computational Results

In this section, we compare the closed-form ZACF formulation to the alternative formulations which include the closed form solutions RACF, TEM, and RATEM, as well as the numerical proximal gradient descent method. For ZACF and TEM, the DFT size used is always $N_d = 2N_{x,d} - 1$. For the reduced-aliasing methods (RACF and RATEM), we pad with a number of zeros equal to 10% or 25% of the training image size, and refer to these results as RACF-10% (RATEM-10%) and RACF-25% (RATEM-25%), respectively. For the proximal gradient descent method, we implement an accelerated backtracking line search method [48] to compute the step size t_k. We initialize the filter ($\bar{\mathbf{h}}_0$) as the conventional filter design subjected to the $prox()$ operator. We use a stopping condition

$$\frac{\left|f(\bar{\mathbf{h}}_{k+1}) - f(\bar{\mathbf{h}}_k)\right|}{\left|f(\bar{\mathbf{h}}_k)\right|} < 10^{-10} \tag{4.42}$$

to terminate the optimization. We refer to our implementation as accelerated proximal gradient descent (APGD).

In this experiment, we train a UOTSDF filter using 9 training images from the AT&T/ORL

Face Dataset [43]; we vary the resolution and crop the training images so that they are square. The platform for this experiment was a desktop running MATLAB 2011a with Windows 7, an Intel Core i7-2600 CPU (3.4 GHz), and 16 GB of RAM. We show the results of our experiment in Fig. 4.11. We only compute the closed form ZACF solution for a resolution up to 50×50 as this is near the memory limit of our machine (similarly, we stop computing the TEM solution for resolutions greater than 70×70).

First, note that TEM outperforms ZACF computationally. Note that, while the fastest method is RACF-10%, we have observed that RACF-25% will perform better from a recognition perspective. This also holds true for RATEM-10% and RATEM-25%. However, recognition performance is dependent on the image size and the training set, so experimentation is needed before choosing the amount of padding used for reduced aliasing methods. Note that RACF-10% is faster than RATEM-10%, whereas RATEM-25% is faster than RACF-25%. The reason for this behavior is difficult to characterize. Note that this is largely dependent on the platform and the structure of the matrices that must be inverted in computing the closed form solutions, and, as such, is difficult to characterize. It is important to note that the complexity of the ZACF, RACF, TEM, and RATEM methods grow at a rate faster than that of APGD. In addition, these methods require a large amount of memory for larger pad sizes. Therefore, we prefer APGD (when available) because it is faster for larger images and is more efficient from a memory perspective. In the next section, we show recognition performance on the AT&T/ORL dataset using all of the described methods to illustrate these points further.

4.5 Results

In this section, we present more thorough results on a variety of datasets to illustrate the performance gains realized by ZACFs. We apply them to three types of pattern recognition tasks, namely face recognition, ATR, and eye localization.

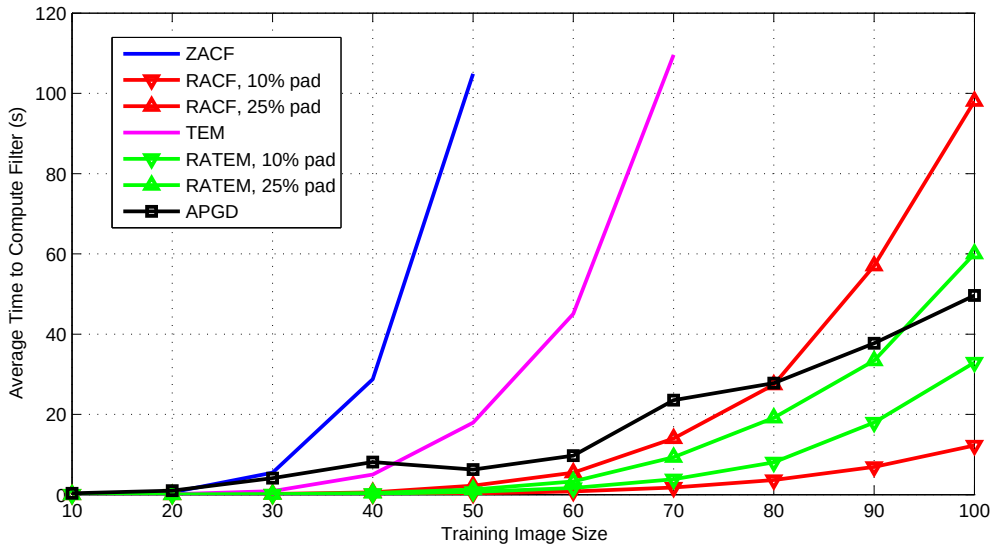


Figure 4.11: Computational times for filter design comparing RACF and the proximal gradient method compared to a closed form solution.

4.5.1 Face Recognition

We apply CFs to two different face recognition datasets: the AT&T/ORL Database of Faces [43] and the face recognition grand challenge (FRGC) dataset [50].

4.5.1.1 AT&T Database of Faces

In the previous chapter, we presented results for the AT&T/ORL Database of Faces using $1/4$ scale images (for computational reasons). Now that we have developed several methods to reduce computational complexity (RACF, TEM, RATEM, and APGD), evaluating filters at full resolution is now possible. However, computing the closed-form solutions (especially the ZACF formulation) is still difficult, and takes many hours on high end server machines with large amounts of memory. Therefore, this dataset is ideal for demonstrating all of the proposed techniques due to the small size of the dataset. Because of this, we have not done an exhaustive comparison on every possible set of parameters available to each filter formulation. Instead, we focus on comparing the filter design techniques themselves.

We use the same procedure as described in the previous chapter with normalized, zero-mean, full resolution (112×92) images. We use PCE as a score metric. We then vary the detection threshold to generate ROC curves, yielding the system's true positive rate and false positive rate for a given threshold. We compute both EER and rank-1 ID rate to serve as metrics for performance.

The results of our validation for the OTSDF, UOTSDF, and MMCF filters are shown in Table 4.2. For each of these filters, we set the tradeoff parameter as $\lambda = 0.9$ (recall that $\mathbf{T} = \lambda\mathbf{D} + (1 - \lambda)\mathbf{P}$). We also set $\lambda_{TEM} = 100$ for TEM filters and use a delta function for the desired correlation output for MMCF. We show results for two different baseline CFs (using different DFT sizes). For the second baseline (DFT of size 223×183), we train the CF and crop the 112×92 upper left section of the template to form the CF template, as performing this cropping generated the best results. We also train CFs using the closed-form expressions for ZACF and TEM, and the numerical method APGD. We observe that the ZACF outperforms both of the baseline CFs. Note that the TEM formulation gives nearly identical performance to that of ZACF, which indicates that it is an acceptable alternative to ZACF. Similarly, APGD gives similar performance to both ZACF and TEM.

We demonstrate the reduced aliasing methods in Tables 4.3 and 4.4 for RACF and RATEM, respectively. In these tables, we have repeated the closed form results for ZACF and TEM, respectively, for convenience. Note that as zero padding increases, performance increases. Also note that the results for RACF mirror closely those of RATEM, once again indicating that each of these techniques are essentially equivalent. Therefore, the ZACF/RACF and TEM/RATEM methods complement each other nicely, as they perform the same from a recognition performance. Depending on the system architecture, one may hold a computational advantage over the other (see again Fig. 4.11). Despite this, we still prefer APGD, as it is the most stable method and is the most memory efficient, as it does not require large matrices to be generated. From this point forward, we will typically use the closed form ZACF formulation if the filter size is small enough; if not, we use APGD to generate the CFs.

Table 4.2: Comparison of Baseline CFs and ZACF Alternatives on the AT&T/ORL Dataset

		Baseline (DFT 112×92)	Baseline (DFT 223×183)	ZACF (closed form)	TEM (closed form)	APGD
OTSDF	EER	11.96%	10.89%	7.539%	7.526%	7.654%
	Rank-1 ID	73.5%	83.75%	89.5%	89.5%	89.5%
UOTSDF	EER	13.25%	11.75%	7.942%	7.884%	7.936%
	Rank-1 ID	72.5%	84.25%	88.25%	88.25%	88%
MMCF	EER	13.25%	11.73%	8%	8%	n/a
	Rank-1 ID	72.5%	84.25%	88.5%	88.75%	n/a

Table 4.3: Comparison of ZACFs and RACFs on the AT&T/ORL Dataset

		ZACF (closed form)	RACF ($p = 15$)	RACF ($p = 20$)	RACF ($p = 25$)	RACF ($p = 30$)
OTSDF	EER	7.539%	11.75%	8.449%	7.212%	7.5%
	Rank-1 ID	89.5%	70.75%	81%	85.75%	87.75%
UOTSDF	EER	7.942%	12.25%	9.25%	8.75%	8.5%
	Rank-1 ID	88.25%	72.75%	83%	85.75%	87.5%
MMCF	EER	8%	12%	8.930%	8.75%	8.5%
	Rank-1 ID	88.5%	73%	83%	86%	87.5%

Table 4.4: Comparison of TEM CFs and RATEM CFs on the AT&T/ORL Dataset

		TEM (closed form)	RATEM ($p = 15$)	RATEM ($p = 20$)	RATEM ($p = 25$)	RATEM ($p = 30$)
OTSDF	EER	7.526%	11.75%	8.442%	7.186%	7.5%
	Rank-1 ID	89.5%	70.75%	81%	85.75%	87.75%
UOTSDF	EER	7.884%	12.218%	9.25%	8.75%	8.5%
	Rank-1 ID	88.25%	72.75%	83%	85.75%	87.5%
MMCF	EER	8%	12%	8.936%	8.75%	8.5%
	Rank-1 ID	88.75%	73%	83%	86%	87.5%

Table 4.5: Comparison of Baseline CFs and ZACFs on the FRGC Dataset

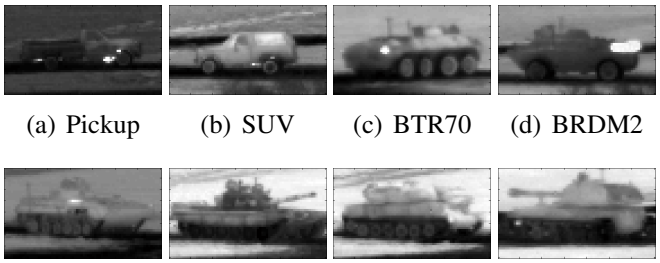
	UOTSDF	ZAUOTSDF
EER	7.346%	5.017%
Rank-1 ID	86.87%	93.78%

4.5.1.2 FRGC

The FRGC dataset [50] contains face images of resolution 128×128. We use 410 subjects from the test portion of the data, removing subjects with less than 8 images per subject. For the data we use, the number of images from each subject varies from 8 (minimum) to 88 (maximum) with a mean of 39 images per subject. We form three training and test sets by randomly selecting 25% of each class for training with the remaining 75% used for testing. We then build one CF per subject and apply every CF to every test image. There are $11,853$ test images and $4,859,730$ correlations per train/test set. Like the AT&T/ORL dataset, we present our results in terms of EER and Rank-1 ID rate in Table 4.5. We choose the UOTSDF filter to evaluate this dataset. Note that the zero-aliasing UOTSDF (ZAUOTSDF) filter (computed via APGD) achieves both a higher Rank-1 ID and a lower EER than the baseline UOTSDF filter. These results are statistically significant. For example, the EER for the UOTSDF filter has a 95% confidence interval of $\pm 0.014\%$ and the EER for the ZAUOTSDF filter has a 95% confidence interval of $\pm 0.011\%$. Because these confidence intervals are so small, we do not show them for these and the rest of the results.

4.5.2 ATR Algorithm Development Image Database

We next investigate vehicle recognition (i.e., simultaneous classification and localization) using a set of infrared images (frames from videos) from the *ATR Algorithm Development Image Database* [51]. This database contains infrared videos (resolution 512×640) of eight military vehicles (one per video). We show these eight classes in Fig. 4.12. Note that some of the vehicles have very similar appearance, making discrimination challenging. In the database, these vehicles



(a) Pickup (b) SUV (c) BTR70 (d) BRDM2

(e) BMP2 (f) T72 (g) ZSU23-4 (h) 2S3

Figure 4.12: Example images of the different classes of military vehicles.

Figure 4.13: Target "pickup" and background

are driven in a circle with diameter 100 m, and therefore exhibit 360° of azimuth rotation. Each video is 1 minute long, allowing the vehicle to complete at least one full circle.The videos in the database are taken at multiple ranges during day and night at 30 fps. We use videos collected during daytime at a range of 1000 m. An example frame is shown in Fig. 4.13. Note that the low quality frame and the general background makes the recognition task challenging.

We design one CF per vehicle to classify the vehicle in the presence of 360° azimuth variation. We select 20 positive-class images per filter (manually cropped from the corresponding frames, of size 40×70) and 80 non-overlapping background images as negative class images for training. We select 200 full frames for testing, which were not a part of the training set. For testing, we

Table 4.6: CF Recognition without Retraining, ATR Algorithm Development Image Database

	Classification		Localization		Recognition	
	Base	ZA	Base	ZA	Base	ZA
MACE	40.0%	51.9%	84.1%	88.9%	36.2%	47.5%
OTSDF	53.6%	62.1%	90.3%	90.3%	52.4%	57.4%
MOSSE	29.7%	32.3%	64.5%	78.1%	26.1%	30.6%
MMCF	51.1%	57.0%	87.7%	90.0%	49.6%	52.6%

correlate the 8 templates with each test image. For each correlation plane, we select the highest value, compute PCE, and assign the test image to the class that gives the highest PCE. If it is the correct class, we declare the video as correctly classified. We declare correct localization when the location of the peak from the true class filter is within 20 and 35 pixels (i.e., half the size of the template) of the ground truth location in the vertical and horizontal directions, respectively. Finally, we declare a correct recognition when there is *both* correct classification *and* correct localization.

Table 4.6 shows the average classification, localization, and recognition percentages of both the conventional CF and ZACF. In these experiments, our baseline CF uses a DFT the size of the training images, as this yielded the best performance. Dalal and Triggs [52] proposed correlating the CF template with the full training frames and adding the subsequent false positives to the training set as negative class training images. The CF is then retrained using this expanded training set. We show our results after retraining in Table 4.7. We observe that retraining helps all filters, but especially the MMCF filter, whose constraints allow unlimited number of training images from two classes. The most important observation is that ZACFs always performs better than or about the same as traditional CFs in classification, localization, and recognition. Note that in these experiments, we treat each frame independently of other frames and do not use a tracker to improve performance. Therefore, the focus is on the performance differences between the original CF formulation and the ZACF formulation.

Table 4.7: CF Recognition with Retraining, ATR Algorithm Development Image Database

	Classification		Localization		Recognition	
	Base	ZA	Base	ZA	Base	ZA
MACE	46.2%	57.7%	85.1%	89.1%	42.7%	51.9%
OTSDF	59.9%	62.3%	90.7%	90.0%	58.1%	58.1%
MOSSE	31.9%	33.4%	76.5%	84.4%	31.1%	32.0%
MMCF	63.9%	74.3%	95.3%	96.1%	63.2%	73.5%

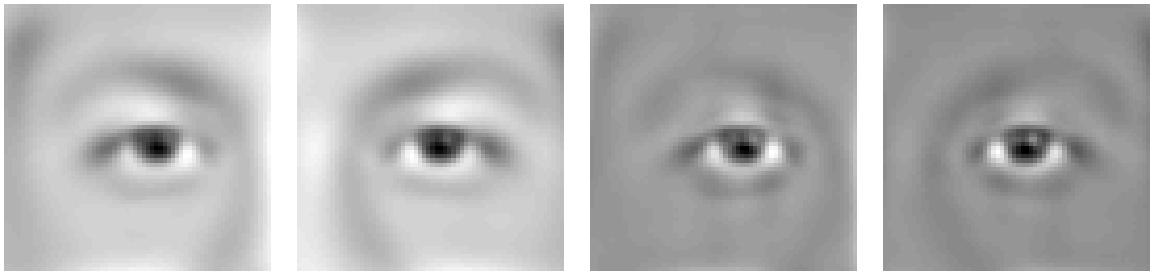
4.5.3 Eye Localization

We now investigate the use of ZACFs for eye localization in face images, which is an important task in face, ocular, and iris recognition. In this experiment we consider the task of localizing the left and right eyes of a detected face. Since a good face detector makes eye localization overly simple, we make the problem more challenging by introducing errors in face localization as outlined in [20]. First, we center the faces obtained using the *OpenCV* face detector [53] to produce 128×128 images with the eyes centered at $(32.0, 40.0)$ and $(96.0, 40.0)$. We then apply a random similarity transform with translation of up to ± 4 pixels, scale factor of up to 1.0 ± 0.1, and rotations of up to $\frac{\pi}{16}$ radians. We used the FERET [54] database, which has about 3400 images of 1204 people. We randomly partitioned the database with 512 images used for training, 675 for parameter selection by cross-validation, and the rest for testing.The CFs are compared by evaluating the normalized distance defined as follows,

$$D = \frac{\left\| P - \hat{P} \right\|}{\|P_l - P_r\|} \tag{4.43}$$

where P is the ground truth location, $\hat{P}$ is the predicted location, and P_l and P_r are the ground truth locations of the left and the right eye, respectively. The point $D = 0.1$ corresponds to detecting an object that is approximately the size of a human iris.

We train MOSSE and ZAMOSSE filters using 64×64 image patches centered at the left and right eye regions. The resulting CF templates are shown in Fig. 4.14. We evaluate the eye localization performance of the CF templates by searching over the entire face image i.e., the



(a) MOSSE (right) (b) MOSSE (left) (c) ZAMOSSE (right) (d) ZAMOSSE (left)

Figure 4.14: Resulting correlation templates for the left and right eyes.

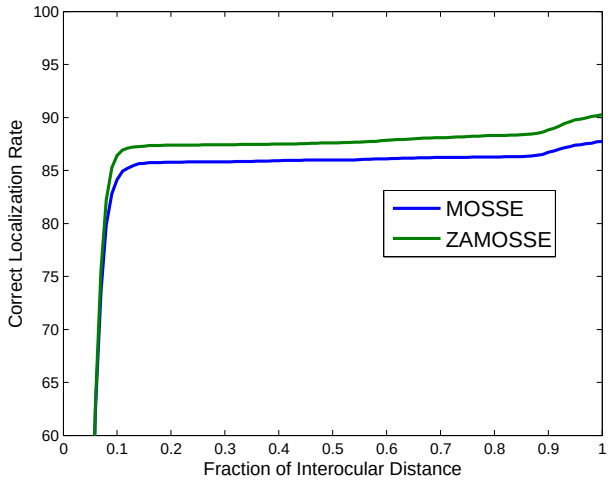
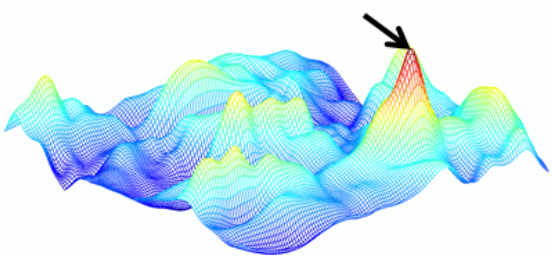


Figure 4.15: Eye localization performance for MOSSE and ZAMOSSE filters.

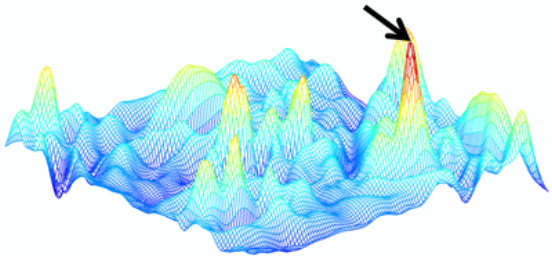
scenario where the approximate eye location is not known apriori. We average the results over 5 different runs with random partitions for training and testing and random similarity transforms. We compare the eye localization performance as a function of D in Fig. 4.15. The filters, while producing a strong response for the correct eye, are sometimes distracted by the wrong eye or other parts of the face. This is because the filters have not been designed using the entire face image. Note that the ZAMOSSE filter provides a higher localization accuracy than the MOSSE filter. At the operating point $D = 0.1$, the MOSSE filter has a localization accuracy of 84.14% and the ZAMOSSE filter has a localization accuracy of 86.39%.

We show example correlation outputs for a test face image in Fig. 4.16. Notice that the ZAMOSSE filter yields a sharper main correlation peak than that produced by the MOSSE filter.

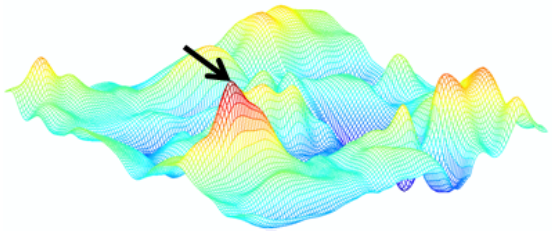
(a) Test Face Image



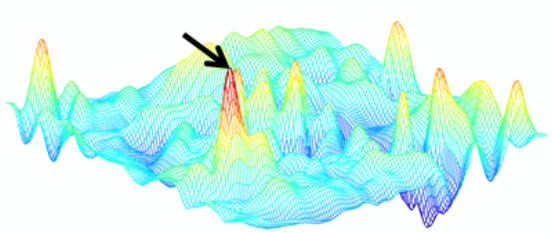
(b) MOSSE



(c) ZAMOSSE



(d) MOSSE



(e) ZAMOSSE

Figure 4.16: Example correlation outputs with the correct peak marked. The test image is shown in (a). Correlation planes for the left eye are shown in (b,c) and for the right eye in (d,e).

This sharper peak translates into better localization performance.

4.6 Conclusions and Discussion

In this chapter, we have introduced several formulations for ZACFs that perform better from a computational perspective. These techniques have allowed for the design of ZACFs using larger training images. As a result, we were able to train and test ZACFs on a variety of datasets and have shown that ZACFs outperform conventional CFs. We summarize the contributions of this

chapter below.

- We introduce RACFs, which use smaller-sized DFTs paired with zero-aliasing constraints to reduce aliasing substantially. We show that this method outperforms using a full-sized DFT and constraining only a portion of the filter tail.
- We introduce TEM, which minimizes the energy of the tail of a correlation template during the design phase. We extended this idea to 2D and illustrate that training the CF is substantially faster than the ZACF formulation. In addition, the TEM formulation easily extends to various CF formulations, which is a key advantage over the ZACF formulation.
- We introduce RATEM, the reduced-aliasing version of TEM, which yields the same filter as the RACF formulation.
- We extend the proximal gradient method (APGD) to ZACFs to numerically solve for the CF templates. We have developed this technique for both constrained and unconstrained CFs.
- We have presented a computational comparison of ZACF, RACF, TEM, RATEM, and APGD. APGD is the preferred method, as it features low memory usage and computational complexity.
- We present an evaluation on the AT&T/ORL dataset of ZACF, RACF, TEM, RATEM, and APGD to show the efficacy of these methods. We show that TEM and RATEM generate the same results at ZACF and RACF, respectively.
- We present results on face recognition, automatic target recognition, and eye localization to show that ZACF methods as a whole outperform conventional CF formulations for localization, classification, and recognition.

In the previous chapters, we have introduced the formulation issue with conventional CF designs. We initially formulated ZACF as a solution to this problem. However, the computational nature of ZACFs make them impractical for some applications. The alternate formulations presented in

this chapter offer additional solutions to the conventional CF formulation problem. In the next part of this thesis, we explore ways to reduce the computational complexity and memory required to apply CFs to test data.

Chapter 5

Overlap-Add and Overlap-Save For Multidimensional Correlations

In the previous three chapters, we introduced the aliasing issues present in CFs and presented several solutions to this problem. In this chapter, we focus on a different problem altogether. We assume we have already designed a CF, and now must apply it to test data in an efficient manner. Note that the work presented in this chapter is therefore applicable to any filtering operation. We focus on the the well-known overlap-add (OLA) and overlap-save (OLS) algorithms [1, 42] and their application to multidimensional correlation.

5.1 Introduction

Correlation and convolution are prevalent techniques in signal processing. Correlation is used to match two signals to each other, whereas convolution is used to filter a signal. These operations are often implemented efficiently in the frequency domain, where the operation is reduced to element-wise multiplication of the DFTs of the two signals. This is a well known technique in signal processing, but is not always implemented in the vision community for filtering applications, as shown recently in [55]. While frequency domain correlation and convolution are

much more efficient than the spatial domain approach (i.e. a sliding window), it can still be very computation and memory intensive, especially when one signal is very different in size from the other. In the case of high-dimensional data (e.g., video) the conventional method may be impossible to implement due to memory constraints. For example, CFs have recently been investigated for the detection of human actions in videos [2, 3, 5, 6]. In this case, the CF is a 3D template that is compared to a larger 3D array (the test video) by sliding it to all possible locations. This operation is typically carried out in the frequency domain, but memory constraints can make this conventional frequency domain approach impossible. Therefore, we look to the OLA and OLS algorithms to provide a more practical solution.

The OLA and OLS algorithms are well-known signal processing techniques. Recently, research has focused on improvements to these techniques. In [56], real signals are processed using OLA and OLS by combining consecutive signal sections as the real and imaginary components of the input signal to the filter. This improves the runtime speed by approximately a factor of 2, but results in an additional delay in the output signal. This is a common trick to use for FFT-based filtering applications, but to simplify our analysis, we will not consider it. In [57], an alternative version of OLA is proposed that sections both the input signal and the filter. This is particularly useful for filters with long impulse responses. In [58], the general form of block digital filtering is generalized to matrix algebra expressions. This analysis is applied to the OLA and OLS implementations in [59], which focuses on efficient and accurate implementations of digital filters that have an impulse response longer than the block size. This problem differs from our application (where the filter is fixed), as we optimize the correlation architecture, not the filter.

In general, most literature focuses on OLA and OLS applied to only 1D. One exception is [60], which considers OLA for block matching for images and evaluates the algorithm from a computational perspective. The work in this thesis differs from [60] in the following ways. First, we offer a theoretical computational analysis for a generic dimension D for both OLA and OLS, whereas [60] only offers a theoretical analysis for OLA in 1D. Furthermore, the analysis in [60]

does not account for several factors in the correlation algorithm complexity. In addition to this, we offer memory insights for both 2D and 3D applications, whereas [60] does not. Finally, we provide insights regarding parameter selection for OLA and OLS given the sizes of the filter template and the signal, whereas [60] only shows that optimal parameters exist. We have presented a 2D analysis of OLA and OLS in [61].

In this chapter, we describe the OLA and OLS algorithms. We then derive general expressions for the computational complexity of both OLA and OLS for a generic D-dimensional signal, and perform an experimental validation of these expressions. We then offer insight to parameter selection so as to minimize the computational time. Finally, we discuss the memory required for each algorithm, and implementation issues for the two and 3D cases.

It should be noted that the computation and memory analysis in this chapter is presented at a high level and does not consider the intricacies of specific hardware platforms, e.g., central processing units (CPUs), graphics processing units (GPUs), or field-programmable gate arrays (FPGAs). It is outside the scope of this thesis to consider specific hardware details, such as considerations for reading and writing to memory. Nevertheless, the analysis presented here should give the reader a high-level overview of the tradeoffs surrounding OLA and OLS, and could serve as a starting point for further, in-depth analysis for particular hardware architectures.

5.2 Description of Algorithms

Both correlation and convolution are similar operations which are used for a wide variety of applications in signal processing, communications, and pattern recognition. Because our interest in this thesis is in CFs, from this point forwards we will, without loss of generality, assume we are performing correlation rather than convolution. We emphasize that our analysis is equally applicable to convolution operations.

A correlation operation is used to measure the similarity between a D-dimensional signal m and a D-dimensional template h, which is shorter than m in each dimension. Correlation may

be expressed as

$$g(x_1, x_2, ...x_D) = m(x_1, x_2, ...x_D) \otimes h(x_1, x_2, ...x_D)$$
$$= \sum_{l_1} \sum_{l_2} ... \sum_{l_D} m(l_1, l_2, ...l_D) h(l_1 + x_1, l_2 + x_2, ...l_D + x_D) \tag{5.1}$$

where $\otimes$ denotes correlation. Note that this can also be written as

$$g(x_1, x_2, ...x_D) = m(x_1, x_2, ...x_D) \star h(-x_1, -x_2, ... - x_D)$$
$$= \mathcal{F}^{-1} \left\{ \mathcal{F} \left(m(x_1, x_2, ...x_D) \right) \odot \mathcal{F}^* \left(h(x_1, x_2, ...x_D) \right) \right\} \tag{5.2}$$

where $\star$ denotes convolution, $\odot$ denotes element-wise multiplication, $\mathcal{F}$ and $\mathcal{F}^{-1}$ denote the DFT and IDFT, respectively, and * denotes complex conjugate. Therefore, correlations may be efficiently implemented using multidimensional DFTs, using the FFT. This results in a large speedup over a sliding window approach. However, it is not efficient when the size of h in the dth dimension, $N_{h,d}$ is considerably smaller than the size of m in the dth dimension, $N_{m,d}$. This is because both h and m must be zero-padded to a size $N_{c,d} \geq N_{h,d} + N_{m,d} - 1$ in the dth dimension. Typically, it is efficient to use a Fourier transform that is a power of 2, so we would typically pad both the signal and the template to the size

$$N_{c,d} = 2^{P(N_{h,d} + N_{m,d} - 1, q_d)} \tag{5.3}$$

in the dth dimension, where the function

$$P(a, q) = x + q - 1,\ x = min(\mathbb{N}) \mid 2^x \geq a \tag{5.4}$$

returns the power of 2 $q \geq 1$ steps greater than a. Here, $\mathbb{N}$ is the set of natural numbers $\{1, 2, 3, ...\}$. For example, if $a = 50$, $2^{P(a,1)} = 64$ and $2^{P(a,2)} = 128$. Therefore, if $N_{h,d}$ is very small compared to $N_{m,d}$, it must be padded with a large number of zeros before the DFT is

Table 5.1: Details of OLA and OLS

	Cut Range	Correlation Range	Final Output Range
OLA	$[k_d L_d + 1, k_d L_d + L_d]$	$[1, N_d]$	$[k_d L_d + 1, k_d L_d + N_d]$
OLS	$[k_d L_d + 1 - (N_{h,d} - 1), k_d L_d + L_d]$	$[N_{h,d}, N_d]$	$[k_d L_d + 1, k_d L_d + L_d]$

computed. This DFT is much larger than the original size of the template. This can be inefficient from both a computational and memory perspective.

We refer to the above method as the conventional method. By contrast, the OLA and OLS methods have been developed to reduce the computational burden of correlations (or to allow for real-time processing of time-domain signals). The OLA and OLS methods are summarized in Table 5.1 and described in the following sections. The index k_d denotes the section number in the dth dimension, and varies from 0 to $K_{OLA,d} - 1$ or $K_{OLS,d} - 1$ as appropriate. The cut range refers to the indices of the input signal (in dimension d) that are processed, and the correlation range specifies the indices of each output correlation that are assigned to the corresponding final output range indices.

5.2.1 Overlap-Add

OLA is based on the concept of dividing the input signal m into small, non-overlapping sections, and computing several correlations with the template h. These correlations are then carefully overlapped and added together to form the final correlation output.

First, the FFT size N_d for dimension d must be chosen such that $N_d > N_{h,d}$. Typically, N_d is a power of 2, and is set as

$$N_d = 2^{P(N_{h,d}, q_d)} \tag{5.5}$$

(Note that selection of the parameters q_d will be discussed later). Then, the length of each section (in dimension d), which we refer to as the cut length, L_d, is determined as

$$L_d = N_d + 1 - N_{h,d} \tag{5.6}$$

This means that sections of the signal measure $L_1 \times ... \times L_d$. Each section is zero padded to size $N_1 \times ... \times N_d$. Next, the number of sections in dimension d is computed as

$$K_{OLA,d} = \left\lceil \frac{N_{m,d}}{L_d} \right\rceil \tag{5.7}$$

This means that the total number of sections will be

$$K_{OLA}^{\#} = \prod_{d=1}^{D} K_{OLA,d} \tag{5.8}$$

For OLA, the output of the correlation for the k_dth section in the dth dimension is assigned to the output range $[k_d L_d + 1,\ k_d L_d + N_d]$. However, note that the output from the next iteration k_{d+1} will overlap with the output from the iteration k_d by $N_d - L_d = N_{h,d} - 1$ samples in the dth dimension. These values are added together (hence the name overlap-add). This process is repeated for each section until the entire correlation plane is computed.

5.2.2 Overlap-Save

OLS is similar to OLA, except the input signal m is divided into *overlapping* sections. Portions of the resulting correlations are then concatenated to form the final correlation output. The FFT size N_d and the cut length L_d are determined as in OLA. For OLS, however, each input section overlaps with the previous section. To determine the required number of sections to process, we find the value for when the iteration *after the final iteration* ($k_d = K_{OLS,d}$) has a final output start index (see Table 5.1) greater than the maximum desired correlation index[1], $N_{h,d} + N_{m,d} - 1$, i.e.

$$K_{OLS,d} L_d + 1 > N_{h,d} + N_{m,d} - 1 \tag{5.9}$$

[1]Note that this corresponds to the output of the conventional correlation scheme. In this case, we compute the full correlation plane, meaning we retain values for which the template only partially overlaps the signal. This is in contrast to some techiniques that only retain values for fully overlapping correlations (see [60]).

This equation can be solved such that the number of sections in the dth dimension for the OLS method is

$$K_{OLS,d} = \left\lceil \frac{N_{m,d} + N_{h,d} - 2}{L_d} + \Delta \right\rceil \tag{5.10}$$

where Δ is a very small positive number $\ll 1$, whose purpose is to make the function round up when the first term has no fractional component. This means that the total number of sections will be

$$K_{OLS}^{\#} = \prod_{d=1}^{D} K_{OLS,d} \tag{5.11}$$

For OLS, only a portion of the output correlation for each section is mapped to the final output (see Table 5.1). These outputs of consecutive correlations are concatenated to form the final correlation output; no addition is necessary as there is no overlap.

5.3 Computational Comparison

In this section, we compare the conventional correlation approach to OLA and OLS from a computational perspective. In this case, we count the number of complex multiplications required by each algorithm. We present theoretical expressions for generalized D-dimensional signals. We validate these expressions experimentally in Section 5.4.

We first introduce the function $f_M(N)$, which returns the number of complex multiplications for an N point FFT. For example, for a radix-2 algorithm, $f_M(N) = \frac{N}{2} log_2 N$ [42]. In this chapter, we assume a radix-2 algorithm without a loss of generality. Optimized figures for non-trivial multiplications for several FFT algorithms may be found in [62].

5.3.1 Conventional

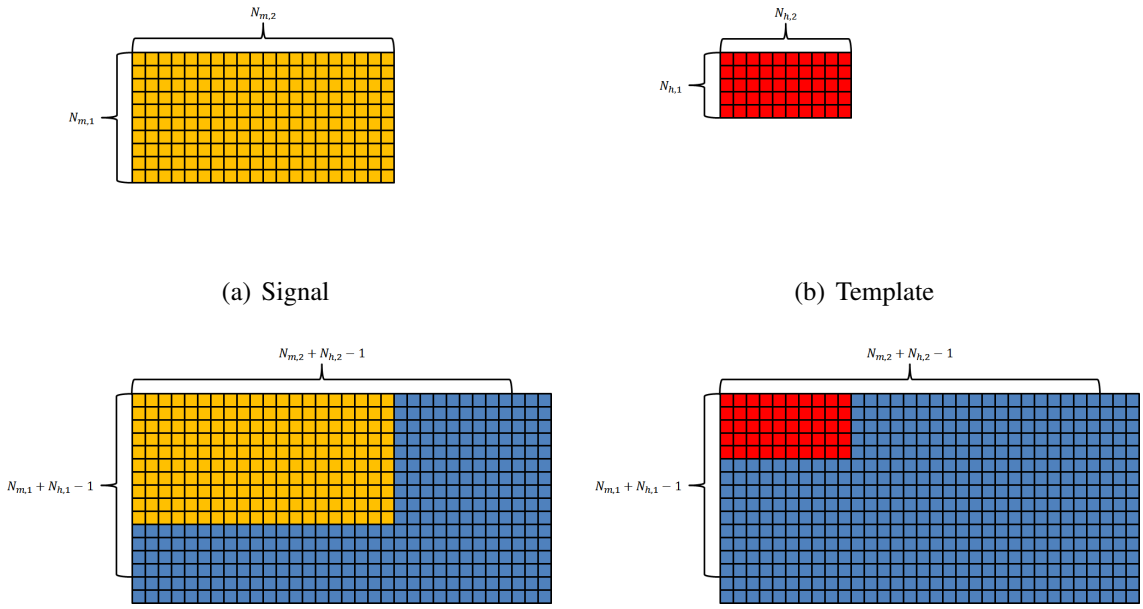
For the conventional approach, in 1D, 3 FFTs are required: one for each of the template, signal, and the product of the two (inverse FFT, or IFFT). In addition, we must account for the element-wise multiplication of the FFTs of both the template and the signal. Therefore, for 1D, the number of complex multiplications is given by

$$C_{conv,1}(N_{c,1}) = 3f_M(N_{c,1}) + N_{c,1} \tag{5.12}$$

In 2D, we must compute multiple FFTs in each dimension. To illustrate this, refer to Fig. 5.1. In this example, we compute the correlation of a signal (Fig. 5.1a) measuring $N_m = 10 \times 20$ and a template (Fig. 5.1b) measuring $N_h = 5 \times 10$. In this case, $N_{m,1} + N_{h,1} - 1 = 14$ and $N_{m,2} + N_{h,2} - 1 = 29$. Using $q_d = 1$, we get $N_{c,1} = 16$ and $N_{c,2} = 32$. Therefore, both the signal and the template are padded to size 16×32 (Fig. 5.1 c and d). The first step is to compute $N_{c,2}$ FFTs of size $N_{c,1}$ for both the template and the signal (the 1D FFT of each column of the padded arrays). Next, $N_{c,1}$ FFTs of size $N_{c,2}$ for both the template and the signal are taken (the 1D FFT of each row). Note that this method is not the most efficient way of computing 2D DFTs (see Appendix D for more detailed discussion) but it will be used for the purposes of this work. Next, the 2D DFT of the template and the 2D DFT of the signal are element-wise multiplied together and the 2D IDFT is computed to generate the output correlation. This can be expressed mathematically as

$$C_{conv,2}(N_{c,1}, N_{c,2}) = 3\left[N_{c,2}f_M(N_{c,1}) + N_{c,1}f_M(N_{c,2})\right] + N_{c,1}N_{c,2} \tag{5.13}$$

Similarly, for a 3D template and signal, the total number of complex multiplications is given by



(a) Signal (b) Template

(c) Padded Signal (d) Padded Template

Figure 5.1: Demonstration of 2D correlation using the conventional method. Both the signal and the template must be padded to size $N_{c,1} \times N_{c,2}$.

$$C_{conv,3}(N_{c,1}, N_{c,2}, N_{c,3}) = 3\left[N_{c,2}N_{c,3}f_M(N_{c,1}) + N_{c,1}N_{c,3}f_M(N_{c,2}) + N_{c,1}N_{c,2}f_M(N_{c,3})\right] + N_{c,1}N_{c,2}N_{c,3} \quad (5.14)$$

We may generalize these results for D dimensions,

$$C_{conv,D}(N_{c,d}) = 3\left[\sum_{d=1}^{D} \frac{f_M(N_{c,d})}{N_{c,d}} N_c^{\#}\right] + N_c^{\#} \quad (5.15)$$

where $N_c^{\#}$ is given by

$$N_c^{\#} = \prod_{d=1}^{D} N_{c,d} \quad (5.16)$$

5.3.2 OLA and OLS

The computational requirements for OLA and OLS are similar, except for the number of sections in each dimension ($K_{OLA,d}$ vs. $K_{OLS,d}$). Here we simply use the variable K_d with the understanding that it is substituted for either $K_{OLA,d}$ or $K_{OLS,d}$. In our notation below, we use the subscript OLX to denote either OLA or OLS.

In 1D, one FFT is required for the template and two FFTs are required for each section (one FFT for the signal section and one IFFT for the inverse transform of the product of the section DFT and the template DFT). In addition, we must account for each of the element-wise multiplications of the template DFT and the signal section DFT. Therefore, for 1D, the number of complex multiplications is given by

$$C_{OLX,1}(N_1, K_1) = (1 + 2K_1) f_M(N_1) + K_1 N_1 \tag{5.17}$$

In 2D, we must compute multiple FFTs in each dimension for each section. The number of FFTs required in one dimension is determined by the size of the section in the opposite dimension. Similarly, the multiplication of the two FFTs is now for a 2D array of values, for each section. For 2D, the total number of complex multiplications is given by

$$C_{OLX,2}(N_1, N_2, K_1, K_2) = (1 + 2K_1K_2) \left[N_2 f_M(N_1) + N_1 f_M(N_2)\right] + K_1K_2N_1N_2 \tag{5.18}$$

As in the conventional case, the method by which the 2D DFTs is computed here is not entirely optimal (see Appendix D for details). However, we will use this method for the purposes of this chapter. In a similar fashion, for 3D, the total number of complex multiplications is given by

$$C_{OLX,3}(N_1, N_2, N_3, K_1, K_2, K_3) = (1 + 2K_1K_2K_3)\,[N_2N_3f_M(N_1) + N_1N_3f_M(N_2) + N_1N_2f_M(N_3)] + K_1K_2K_3N_1N_2N_3 \quad (5.19)$$

We may generalize this result for D dimensions

$$C_{OLX,d}(N_d, K_d) = \left(1 + 2K^{\#}\right)\left[\sum_{d=1}^{D} \frac{f_M(N_d)}{N_d} N^{\#}\right] + K^{\#}N^{\#} \quad (5.20)$$

where $K^{\#}$ is substituted by $K^{\#}_{OLA}$ or $K^{\#}_{OLS}$, as appropriate, and $N^{\#}$ is given by

$$N^{\#} = \prod_{d=1}^{D} N_d \quad (5.21)$$

5.3.3 The OLS Curse of Dimensionality

In this section, we compare OLA to OLS, in terms of complexity. To further simplify expressions, we rewrite Eq. 5.20 as

$$C_{OLX,d}(N_d, K_d) = \left(1 + 2K^{\#}\right)\alpha_F + K^{\#}\alpha_M \quad (5.22)$$

where α_F represents the contribution of the FFT operation to the total complexity,

$$\alpha_F = \sum_{d=1}^{D} \frac{f_M(N_d)}{N_d} N^{\#} \quad (5.23)$$

and α_M represents the contribution of the element-wise multiplications to the total complexity,

$$\alpha_M = N^{\#} \quad (5.24)$$

If we are interested in comparing OLA to OLS, we may determine R, the ratio of the complexity

of OLS to the complexity of OLA,

$$R = \frac{C_{OLS,d}}{C_{OLA,d}} = \frac{\left(1 + 2K_{OLS}^{\#}\right)\alpha_F + K_{OLS}^{\#}\alpha_M}{\left(1 + 2K_{OLA}^{\#}\right)\alpha_F + K_{OLA}^{\#}\alpha_M} \tag{5.25}$$

A likely scenario is to pre-compute the FFT of the template. This would apply to a scenario where we are correlating many signals with the same template. If this is the case, we can simplify Eq. 5.25 to

$$R = \frac{K_{OLS}^{\#}\left(2\alpha_F + \alpha_M\right)}{K_{OLA}^{\#}\left(2\alpha_F + \alpha_M\right)} = \frac{K_{OLS}^{\#}}{K_{OLA}^{\#}} \tag{5.26}$$

To understand this ratio, we must first understand how $K_{OLS,d}$ and $K_{OLA,d}$ relate to each other. We assert that for $N_{h,d} \geq 2$,

$$K_{OLS,d} \geq K_{OLA,d} \tag{5.27}$$

This is not guaranteed for when $N_{h,d} = 1$, but this case is rare and of limited usefulness (i.e. templates are not just one sample long in practice). Therefore, R is always greater than or equal to 1. More interestingly, though, R can oftentimes in practice take on much higher values. For example, if $K_{OLA,d} = 4$ and $K_{OLS,d} = 5$, $R = 1.25$ for $d = 1$, 1.56 for $d = 2$, and 1.95 for $d = 3$. Therefore, while many sources present OLA and OLS as relatively similar algorithms from a computational complexity perspective [1, 42], we show here that OLS is more complex than OLA, and becomes even more complex with increasing dimension. In Section 5.4, we show a case for which $R = 8$ for $d = 3$. This results in a runtime that is longer than even the conventional case.

The difference between OLA and OLS diminishes if we break the input signal into many sections. However, it is often the case in image and video processing that the number of sections in each dimension is lower, and, as a result, there is a significant gap between OLA and OLS. Poor performance for OLS can occur when blindly choosing OLS parameters to perform a correlation.

Note that the above expression for R only holds when OLA and OLS use the same value of N_d; it does not hold when comparing OLA and OLS algorithms that use different values for N_d. It is important to remember that careful selection of N_d (by choosing the proper q_d) is important for both algorithms. We will discuss parameter selection for these algorithms in Section 5.5.

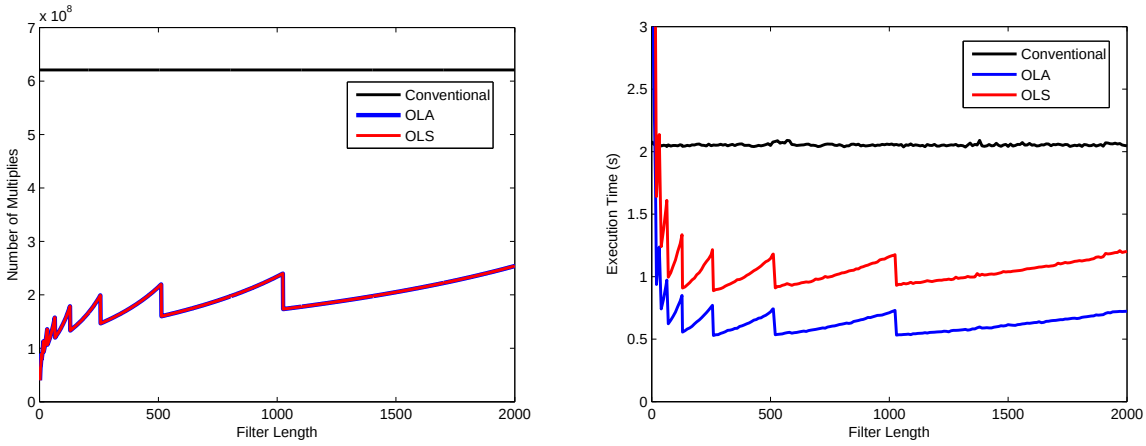
5.4 Experimental Validation

To compare OLA and OLS to the conventional scheme, we implement all three algorithms in 1D, 2D, and 3D. To illustrate this, we used a desktop running MATLAB 2011a with Windows 7, an Intel Core i7-2600 CPU (3.4 GHz), and 16 GB of RAM. Our main goal is to show that the expressions for complexity presented in Sections 5.3.1 and 5.3.2 can serve as an estimate for the actual computational complexity of the implemented algorithms.

5.4.1 1D case

In general, an OLA or OLS scheme must segment the signal into sections which are then transformed via FFT. In our MATLAB implementation, we avoided `for` loops to do this processing, relying instead on vectorized operations. However, there were two exceptions; for the OLS case, a `for` loop was used to partition the input signal, which was necessary because consecutive sections overlap. For the OLA case, a `for` loop was used to add overlapping segments together. This represents a computational overhead that is not accounted for in the equations in the previous sections.

In Fig. 5.2a, we vary the template length for a fixed signal length $N_m = 10^7$, showing the theoretical number of complex multiplications (Eq. 5.12 and 5.17). Note that OLA and OLS are approximately the same here, because N_m is very large and the $K_{OLA} - K_{OLS}$ is small compared to the magnitude of K_{OLA} or K_{OLS}. We show the results of our experimental runtime in Fig. 5.2b. For the conventional case, we use an FFT size that is one size larger than $N_m + N_h - 1$



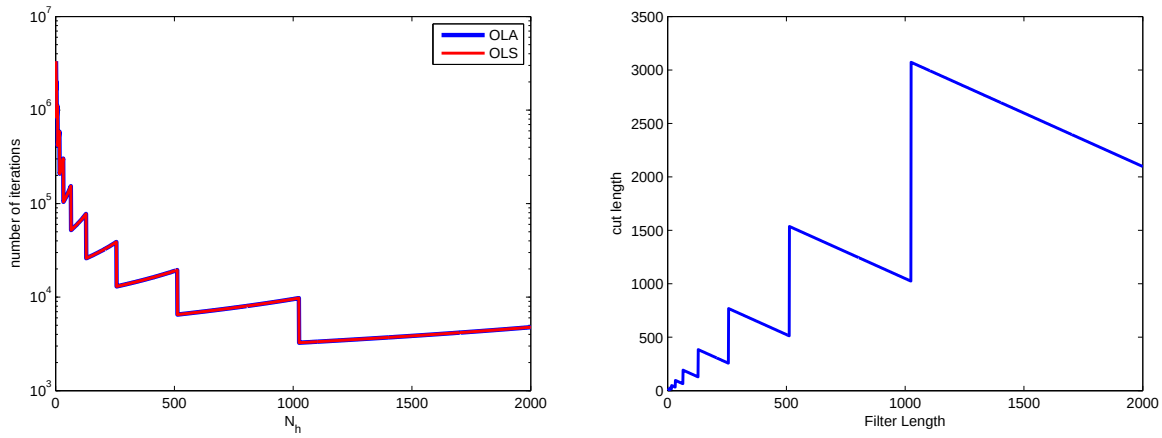
(a) Theoretical

(b) Experimental

Figure 5.2: Theoretical and experimental results for each correlation algorithm for the 1D case. For the plot on the left, note that the OLA curve is difficult to see because it is behind the OLS curve.

(i.e., $q_d = 1$). In the OLA and OLS case, we set $N_d = 2^{P(N_{h,d},q_d)}$, with $q_d = 2$. Note that OLA and OLS in general perform better than the conventional scheme. The experimental results in general match the theoretical results, except for shorter template lengths, which corresponds to a very short cut length and a large number of iterations (see Fig. 5.3). In this case, more time is spent on algorithm overhead (i.e. segmenting the signal and joining the outputs). This is why the experimental case does not match up with the theoretical case, which only accounts for the number of complex multiplications needed for the correlation operations. For smaller values of N_m, this effect is more pronounced, because the correlations necessary are less computationally intensive than the algorithm overhead.

For our particular implementation, we observed that OLA greatly outperforms OLS. This is due in part to two reasons: one, OLA requires less iterations and therefore less multiplications; two, segmenting the input signal for OLA is in general easier to do efficiently, because there is no overlap. When choosing an OLA or OLS algorithm for correlation, the specific system parameters should be taken into account, and the overhead required for processing the signals should not be overlooked. As the correlation computations become larger and more time inten-



(a) Number of Iterations

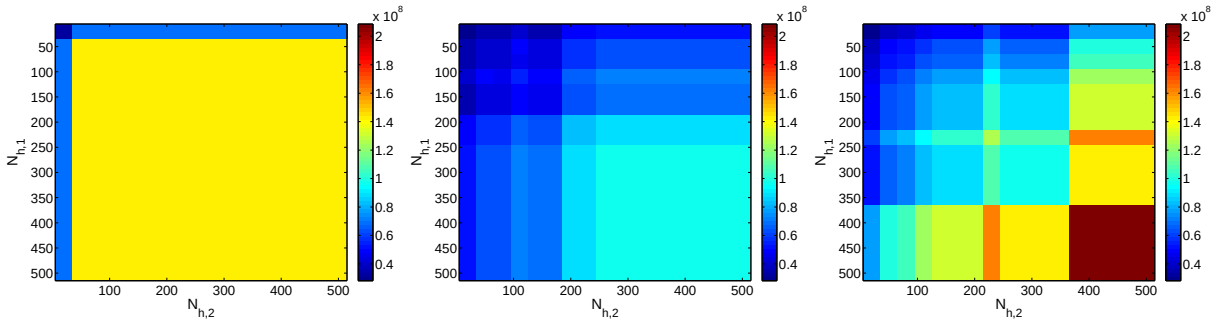
(b) Cut Length

Figure 5.3: Number of iterations (K_{OLA} and K_{OLS} for OLA and OLS, respectively) and cut length (L_d) for the 1D case. For the plot on the left, note that the OLA curve is difficult to see because it is behind the OLS curve.

sive, overhead complexity in general is less important, and takes up a much smaller percentage of the total algorithm runtime.

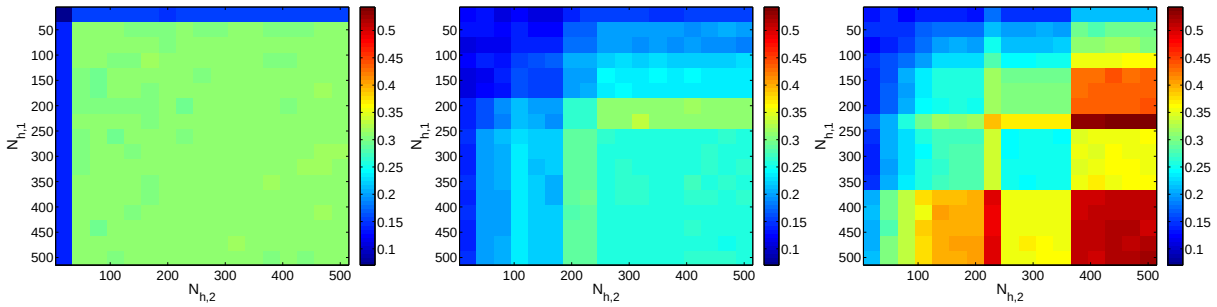
5.4.2 2D Case

To verify the 2D case, we use an image of size $N_m = 1000 \times 1000$, and sweep the template size from $N_h = 20 \times 20$ to 500×500, testing possible template sizes with a step size of 30. We repeat our experiment 10 times and show the average results. We use $q_d = 1$ for the conventional case and $q_d = 2$ for OLA and OLS. The theoretical number of complex multiplications for the conventional, OLA, and OLS schemes are shown in Fig. 5.4. The experimental results are shown in Fig. 5.5. We note that the experimental results match the theory well. We validate both theoretically and experimentally that OLS performs worse than the conventional scheme for larger template sizes, whereas OLA performs approximately the same or better than the conventional scheme for all template sizes. Note that this experiment does not necessarily represent the optimal parameters for OLA and OLS, and is intended to validate our theoretical expressions rather than show that OLA and OLS are better than the conventional scheme.



(a) Conventional (b) OLA (c) OLS

Figure 5.4: Theoretical complexity for correlation algorithms using $N_m = 1000 \times 1000$. Units are number of complex multiplications.



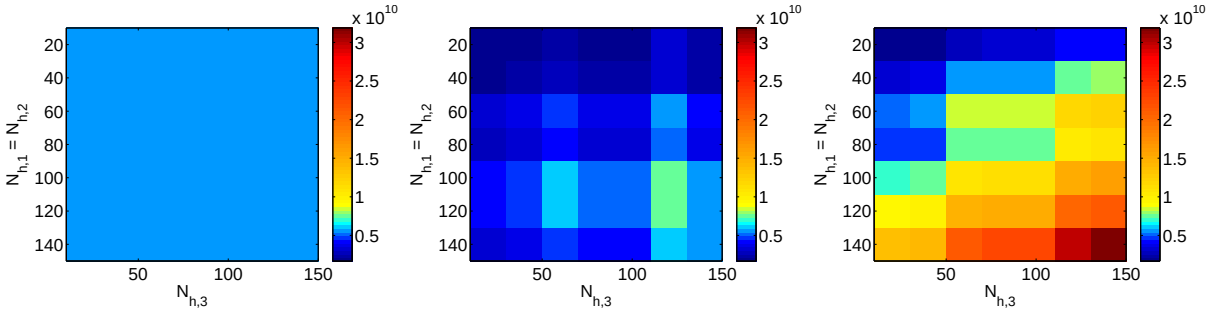
(a) Conventional (b) OLA (c) OLS

Figure 5.5: Experimental runtime for correlation algorithms using $N_m = 1000 \times 1000$. Units are seconds.

5.4.3 3D Case

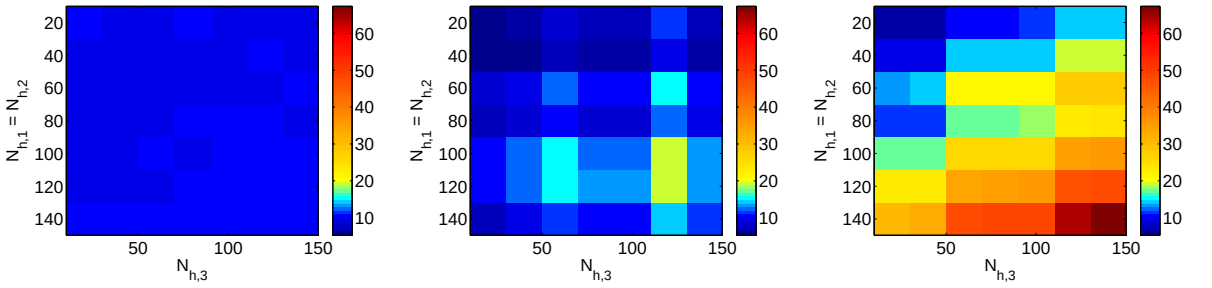
To verify the 3D case, we use a fixed video size of $N_m = 240 \times 320 \times 300$. We use a small video because of memory constraints - the video must be small enough for the conventional scheme to be able to compute the correlation without saturating the system memory, which would affect computation speed. We set the template's dimensions as $N_{h,1} = N_{h,2}$ and vary $N_{h,3}$ independently. The template dimensions are varied from 20 to 140 with a step size of 20. We repeat our experiment 10 times and average the results. We use $q_d = 1$ for the conventional case and $q_d = 2$ for OLA and OLS. We compute the theoretical number of complex multiplications for the conventional, OLA, and OLS schemes and show them in Fig. 5.6. The results of our experiment are shown in Fig. 5.7. Note that our experimental results follow the theoretical predictions well. For these figures, we use a piecewise linear color scaling to help highlight the differences between the conventional scheme and OLA, because the two are very similar to each other and very different from OLS.

We observe that the conventional scheme performs slightly better than expected. However, OLA outperforms the conventional scheme for smaller template sizes. Note that while OLA performs similarly to the conventional scheme, OLS performs much worse. Because OLA and OLS are using the same value of q_d, OLS requires more iterations than OLA, and the curse of dimensionality is in effect (see Section 5.3.3). For example, a template of size $N_h = 120 \times 120 \times 120$ requires 7.75×10^9 complex multiplications for OLA and 2.033×10^{10} complex multiplications for OLS, which is 2.62 times as much. In our experiment, OLA took $19.31\ s$ and OLS took $46.45\ s$, which is 2.41 times longer. In this case, OLA requires $(K_1, K_2, K_3) = (2, 3, 3)$ and OLS requires $(K_1, K_2, K_3) = (3, 4, 4)$. If we had precomputed the template DFT, OLS would take 2.67 times longer for OLS compared to OLA (Eq. 5.26). In our case, the factor is a little lower because our timing numbers include the time to take the DFT of the template. As in the 2D case, it is important to realize the purpose of this section is to validate our theoretical expressions. The parameters used for OLA and OLS here are not necessarily optimal. In the next



(a) Conventional (b) OLA (c) OLS

Figure 5.6: Theoretical complexity for correlation algorithms using $N_m = 240 \times 320 \times 300$. Units are number of complex multiplications.



(a) Conventional (b) OLA (c) OLS

Figure 5.7: Experimental runtime for correlation algorithms using $N_m = 240 \times 320 \times 300$. Units are seconds.

section, we turn our attention to parameter selection for OLA and OLS.

5.5 Parameter Selection

In this section, we discuss parameter selection for OLA and OLS. The primary parameter choice is the FFT size, N_d, which in general is chosen to be a power of 2 greater than $N_{h,d}$. The choice of N_d in turn affects the value of L_d and K_d, where K_d represents $K_{OLA,d}$ or $K_{OLS,d}$ as appropriate. In [60], it is argued that there exists an optimal ratio, r_d, for which complexity is a minimum,

$$r_d = \frac{N_d}{N_{h,d}} \tag{5.28}$$

(note that we have used our own notation for algorithm parameters). It is noted that the number of iterations, K_d, decreases and the complexity for each iteration increases with increasing r. This tradeoff means that there is an optimal $\hat{r}_d$. However, we note that Eq. 15 and 17 in [60] fail to account for both the FFT and IFFT needed to perform each correlation (instead, only one is accounted for). In addition, [60] does not account for the element-wise multiplications of the FFTs of the signal section and the template. We show that these element-wise multiplications should not be ignored, as they account for a significant portion of the complex multiplications for each correlation operation, particularly for lower values of N_d.

To illustrate this concept, we rewrite α_F and α_M (originally Eq. 5.23 and 5.24) in terms of r_d [2]

$$\alpha_F = \sum_{d=1}^{D} \frac{f_M(r_d N_{h,d})}{r_d N_{h,d}} r^{\#} N_h^{\#} \tag{5.29}$$

$$\alpha_M = r^{\#} N_h^{\#} \tag{5.30}$$

where

$$r^{\#} = \prod_{d=1}^{D} r_d \tag{5.31}$$

$$N_h^{\#} = \prod_{d=1}^{D} N_{h,d} \tag{5.32}$$

We may also rewrite Eq. 5.6 as

$$L_d = N_{h,d}(r_d - 1) + 1 \tag{5.33}$$

[2]Note that in our expressions, we correct the forgotten terms in [60] and also generalize to the d-dimensional case. The expressions in [60] are only for 1D. Furthermore, we do not normalize our expressions by the number of output correlation values.

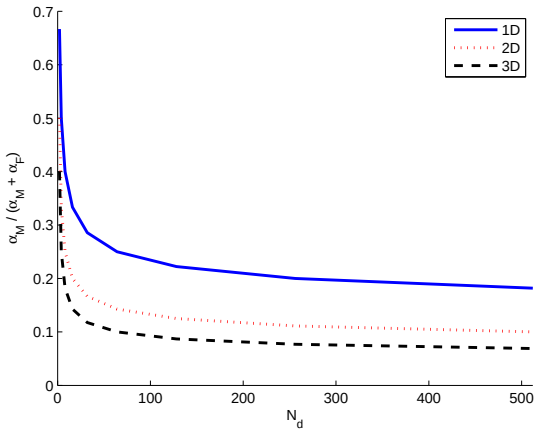
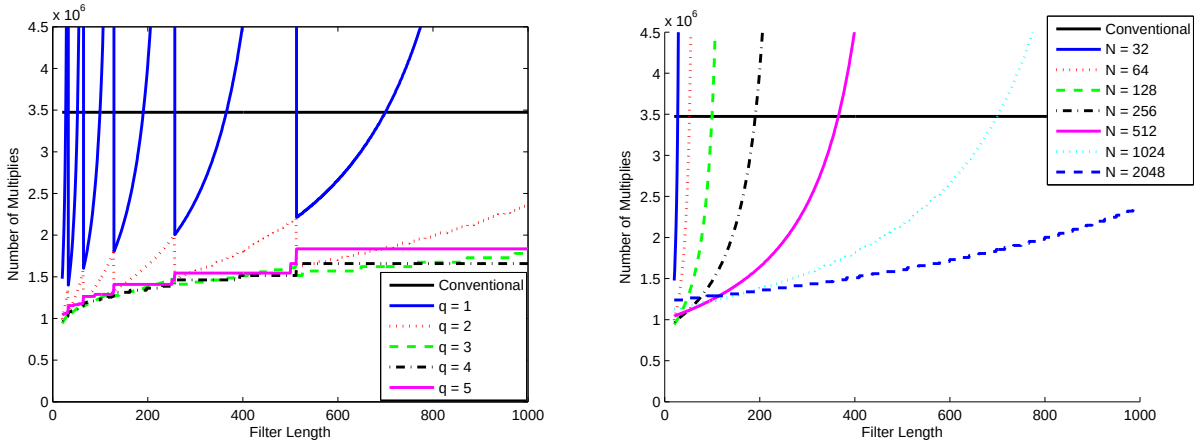


Figure 5.8: Comparison of the magnitudes of the terms in Eq. 5.22.

Notice that both α_F and α_M increase as any r_d increases. On the other hand, as r_d increases, L_d increases which causes $K_{OLA,d}$ (Eq. 5.7) and $K_{OLS,d}$ (Eq. 5.10) to decrease. This is the fundamental tradeoff between number of iterations and complexity per iteration. As mentioned previously, [60] did not include the term α_M in their expression for computational complexity. To illustrate how this term affects the computational complexity, we consider the cases for 1D, 2D, and 3D correlations. In the multidimensional case, we assume that N_d is the same for all d, for simplicity. We plot $\alpha_M/(\alpha_M+\alpha_F)$ for different dimensions. This term represents the relative size of α_M to the sum of α_M and α_F, not accounting for the coefficients in Eq. 5.22. As can be seen in Fig. 5.8, the α_M term carries a significant weight for lower-dimensional signals as well as schemes that use a smaller value of N_d. Therefore, it should be retained in any expression for complexity. Next, we go through each case (1D, 2D, and 3D) and discuss parameter selection to achieve optimal performance.

5.5.1 1D Case

Unfortunately, it is not exactly intuitive to optimize the correlation algorithm by tweaking the values of r (for this and subsequent variables in this section, we drop the subscript d, as all quantities are in 1D). If we sweep the template size (N_h) and hold the FFT size (N) constant, we



(a) Different Values of q_d

(b) Fixed values of N_d

Figure 5.9: Theoretical computational complexity for OLA.

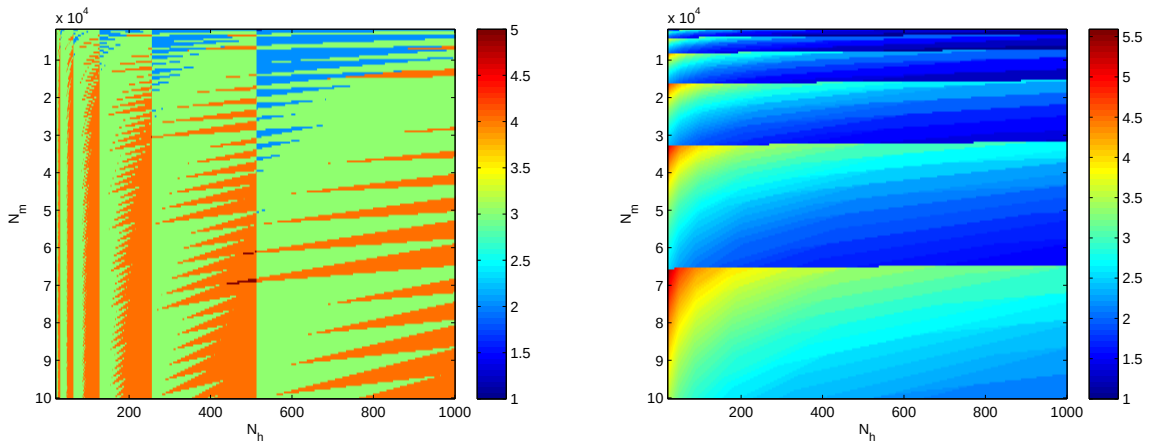
are in effect only changing L. A small template size will result in a large L and therefore a small number of sections, K, which are always processed with N point FFTs. Therefore, for a fixed N, the best performance is always achieved for the smallest template size possible, as this minimizes the number of iterations. The more interesting case is when N changes dynamically with N_h. In this case, r does change, but it only takes on discrete values, as we require N to be a power of 2. Therefore, we sweep the template length for a fixed sequence length for different values of $q \geq 1$ using Eq. 5.5. The computational complexity for OLA for $N_m = 10^5$ is shown in Fig. 5.9a. A similar result holds for OLS. Notice that using $q = 1$ results in poor performance, especially as N_h approaches N (as this happens, the cut length L shrinks to 1 and the number of iterations K approaches N_m for OLA and $N_m + N_h - 1$ for OLS. We point out that each q-curve is actually a cut from a curve for which N is held constant (see Fig. 5.9b). By dynamically changing q as a function of N_h, we are effectively switching from curve to curve. This is particularly obvious in the figures for the curves for $q = 1$ and $q = 2$, but for $q \geq 3$ the method is selecting values of N greater than those shown in Fig. 5.9b. We observe that a value of $q \geq 2$ is acceptable and provides performance better than the conventional scheme; however a value of $q = 3$ or 4 tends to perform the best for the template size ranges shown.

The above discussion holds true for the particular example we used to illustrate the problem. However, if the sequence length changes, then the optimal q will change. If q is too large, the FFT size will approach the size of the FFT used for the conventional approach and we will effectively no longer be doing OLA or OLS (as the signal will not be divided, but rather processed in one section). We therefore vary the sequence length and template length simultaneously and report both the best q and the corresponding speedup factor (compared to the conventional scheme) for each case. In our evaluation we varied N_h from 20 to 1000 and varied N_m from 2000 to 10^5. We present the results for OLA in Fig. 5.10. The results for OLS are visually identical and only vary slightly (for this particular example), and therefore are not shown.

We find that the best value of q alternates between 3 and 4 for longer signals, while $q = 2$ is often optimal for shorter signals. In general, we observe that $q = 2$ exhibits the same problem as $q = 1$, in that the computational complexity rises significantly when N_h approaches a power of 2 (this can be seen in Fig. 5.9). This effect is not as pronounced for $q = 3$ or $q = 4$. In this experiment, compared to the conventional scheme, the maximum observed speedup factor for both OLA and OLS was 5.59 and the average speedup factor was 2.39 for OLA and 2.36 for OLS. We noticed that there were several cases in which the best value of q for OLS actually performed slightly worse than the conventional scheme, whereas the worst-case scenario for OLA achieved identical performance to the conventional scheme.

5.5.2 Multidimensional Cases

For the 2D case, we consider square images and templates, for simplicity. We vary $N_{h,1} \times N_{h,2}$ from 20×20 to 500×500, and we vary $N_{m,1} \times N_{m,2}$ from 600×600 to 1500×1500. In this case, we must select both N_1 and N_2 based on $N_{h,1}$ and $N_{h,2}$, respectively. We do so using Eq. 5.5, where now we have two parameters (q_1 and q_2) for which all possible combinations must be considered (i.e., q_1 may or may not be equal to q_2). For the template and image size ranges above, we report the best combination of q_1 and q_2 and the corresponding speedup factor for



(a) Best q (b) Speedup Factor For Best q

Figure 5.10: The best q_d and corresponding speedup factor for OLA.

OLA (Fig. 5.11) and OLS (Fig. 5.12). Note that the best performance for larger templates is typically achieved for q_d values of 2 or 3, with smaller template sizes performing best, in general, for q_d values between 3 and 5.

We repeat this evaluation in 3D. Similar to the 2D case, we vary $N_{h,1} \times N_{h,2} \times N_{h,3}$ from $20 \times 20 \times 20$ to $500 \times 500 \times 500$, and we vary $N_{m,1} \times N_{m,2} \times N_{m,3}$ from $600 \times 600 \times 600$ to $1500 \times 1500 \times 1500$. In this case, we now report the best combination of q_1, q_2, and q_3 and the corresponding speedup factor for OLA (Fig. 5.13) and OLS (Fig. 5.14). Like the 2D case, we mostly observe good performance for larger templates for q_d values of 2 or 3, with smaller templates performing better for q_d values between 3 and 5.

Finally, we present the minimum, mean, and maximum speedup factors in Table 5.2. Note that for consistency, the numbers for 1D represent the same template and signal size variations as the 2D and 3D cases (N_h varied from from 20 to 500 and N_m varied from 600 to 1500). Both OLA and OLS always share the same maximum speedup factor; however, OLS in some cases (larger template sizes) performs worse than the conventional scheme. This effect is enhanced for higher dimensions (recall the OLS curse of dimensionality, which still manifests itself to an extent when choosing optimal algorithm parameters). Therefore, if OLS is used, care should be

Table 5.2: Comparison of Speedup Factors for Different Dimensions

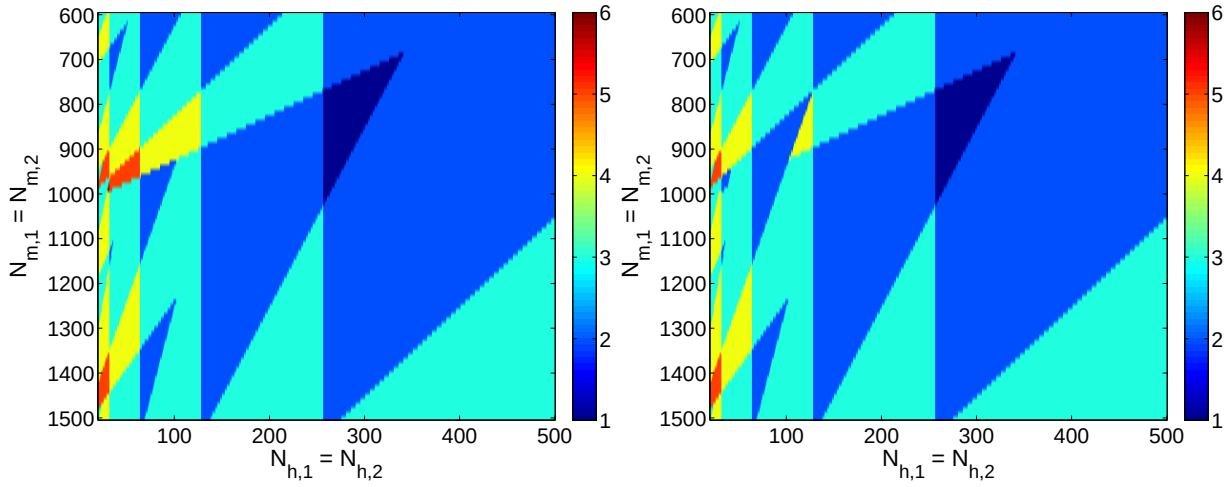
	Min	Mean	Max
OLA, $D = 1$	1	1.43	3.41
OLA, $D = 2$	1	1.67	5.78
OLA, $D = 3$	1	1.90	9.40
OLS, $D = 1$	0.59	1.25	3.41
OLS, $D = 2$	0.33	1.34	5.78
OLS, $D = 3$	0.17	1.45	9.40

taken to avoid this situation to ensure the speedup factor is larger than 1. We also notice that OLA always outperforms or performs the same as the conventional scheme. Finally, we notice that the average and maximum speedup factors increases with dimension, which shows that OLA and OLS in general are more beneficial for higher-dimensional correlations than lower-dimensional correlations. To illustrate the benefit of OLA over OLS further, we show the speedup factor of OLA relative to OLS for 2D and 3D in Fig. 5.15. These plots indicates how many times faster OLA is than OLS. Note that for all template and signal sizes, OLA is either similar or better than OLS. The gap between the two is more significant in higher dimensions.

Note that in all cases, there is no single answer regarding how to choose parameters for OLA and OLS. We have simply illustrated a method by which these parameters may be chosen. For signal and template sizes outside the ranges shown here, a similar analysis would have to be conducted.

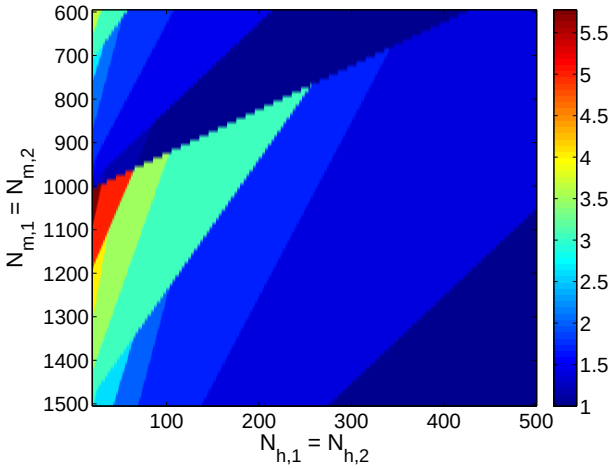
5.6 Memory Analysis for 1D and 2D Applications

We now turn our attention to the memory requirements for the conventional, OLA, and OLS methods. In previous literature on OLA and OLS, memory requirements have typically not been considered. However, with modern correlation methods being applied to high-dimensional data (e.g. large images or videos), memory considerations are of importance. Because OLA and OLS process these signals a section at a time, they have a tremendous advantage over the conventional



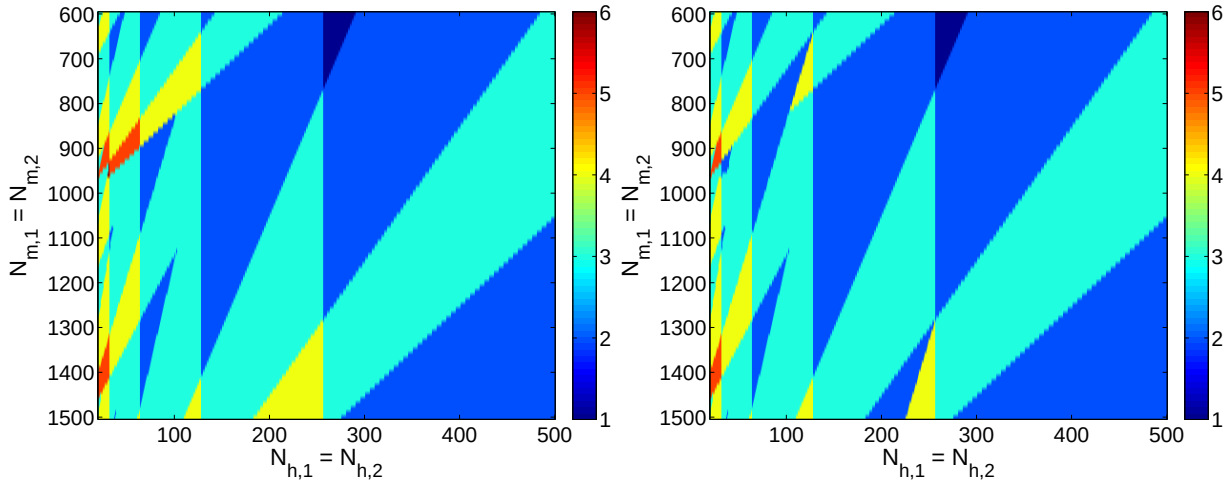
(a) Best q_1

(b) Best q_2



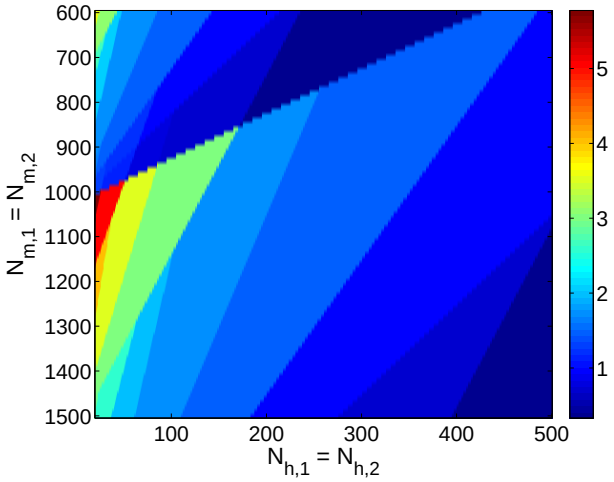
(c) Speedup Factor, Best Combination of q_1 and q_2

Figure 5.11: Parameter selection for 2D OLA.



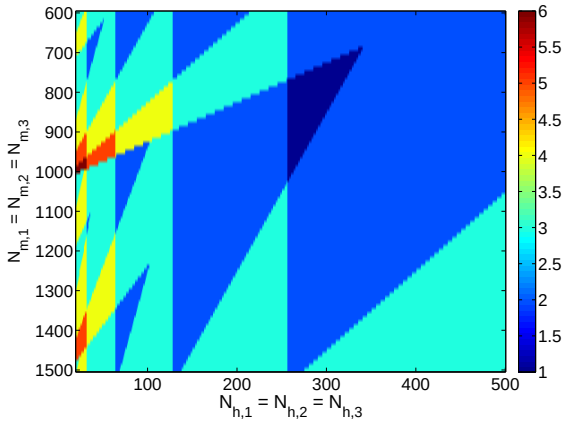
(a) Best q_1

(b) Best q_2

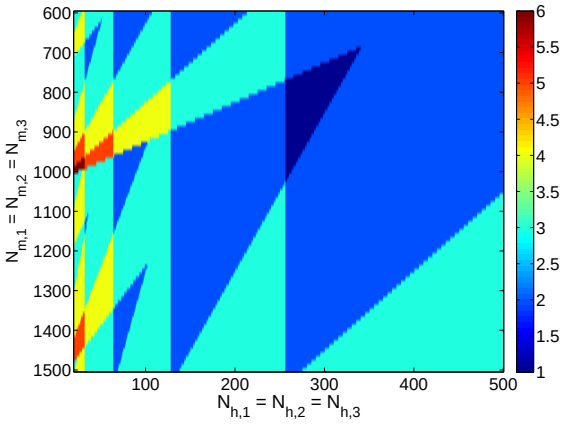


(c) Speedup Factor, Best Combination of q_1 and q_2

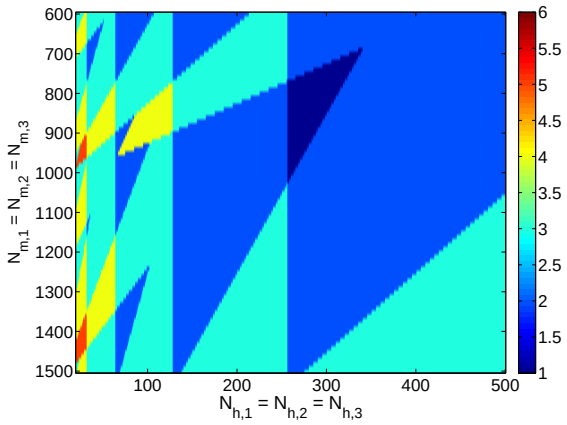
Figure 5.12: Parameter selection for 2D OLS.



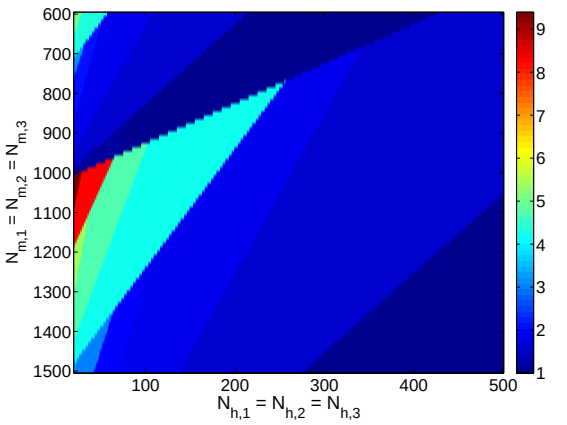
(a) Best q_1



(b) Best q_2

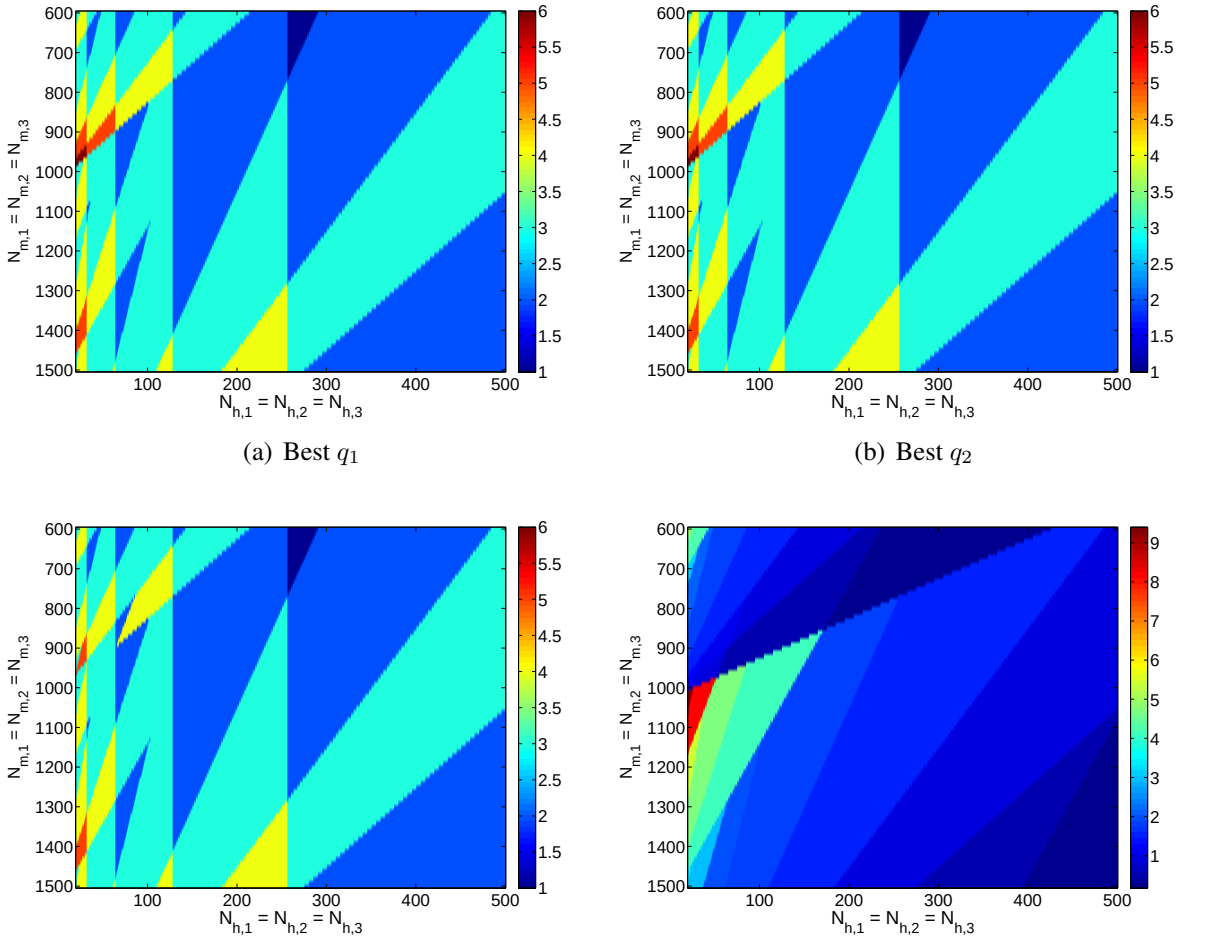


(c) Best q_3



(d) Speedup Factor, Best Combination of q_1, q_2, and q_3

Figure 5.13: Parameter selection for 3D OLA.



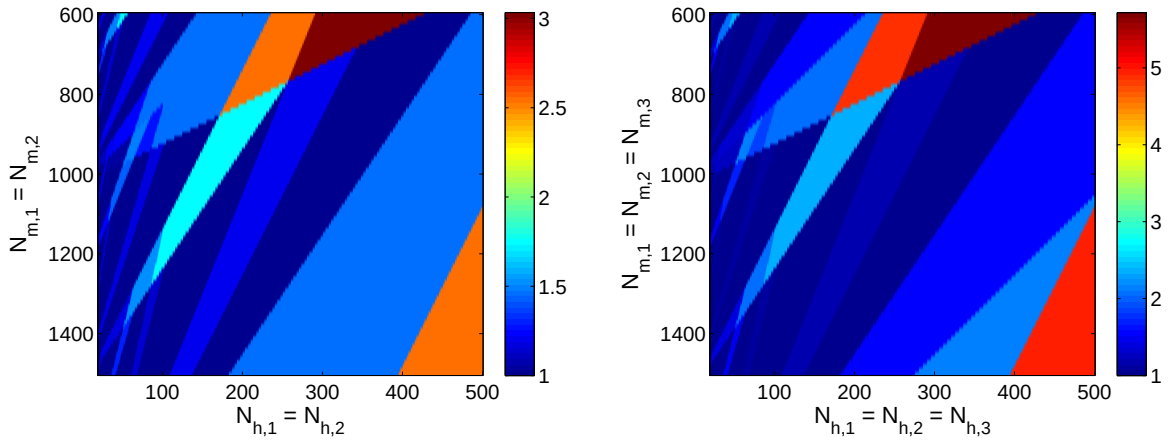
(a) Best q_1

(b) Best q_2

(c) Best q_3

(d) Speedup Factor, Best Combination of q_1, q_2, and q_3

Figure 5.14: Parameter selection for 3D OLS.



(a) 2D Case (b) 3D Case

Figure 5.15: Speedup factor of OLA compared to OLS. The plots indicate how many times faster OLA is compared to OLS.

technique. We also highlight an important memory distinction between OLA and OLS, which we first noted in [61]. In this section, we focus on 2D applications (we also discuss 1D applications, but they are trivial). We investigate 3D applications in Section 5.7.

In this section and the next, we will present equations that represent the memory required for different algorithms. In all cases, the memory indicates the number of elements that must be stored. Each value must be multiplied accordingly by the data type, e.g., for double precision, the memory (in bytes) would be $8E$, where E is an expression for the number of memory elements. For plots and tables, we have assumed double precision and have converted to common memory units (1 kB = 1024 B and 1 MB = 1024 kB, etc.).

For all three methods, the most memory-intensive step of the algorithm is the multiplication in the frequency domain. This is when two complex-valued Fourier transforms are element-wise multiplied to result in a third complex valued result. We assume in-place multiplications. We further assume that unneeded variables may be cleared from memory after they are processed (e.g., the template h may be cleared from memory after we take its DFT).

5.6.1 Conventional Method

For the conventional approach in 1D, we use an FFT size of N_c. Because there are two transforms being multiplied, and because the data is complex, the maximum instantaneous memory usage is $4N_c$ elements. In 2D, the size of the arrays is now determined by both $N_{c,1}$ and $N_{c,2}$ such that the maximum instantaneous memory usage is $4N_{c,1}N_{c,2}$.

5.6.2 OLA and OLS

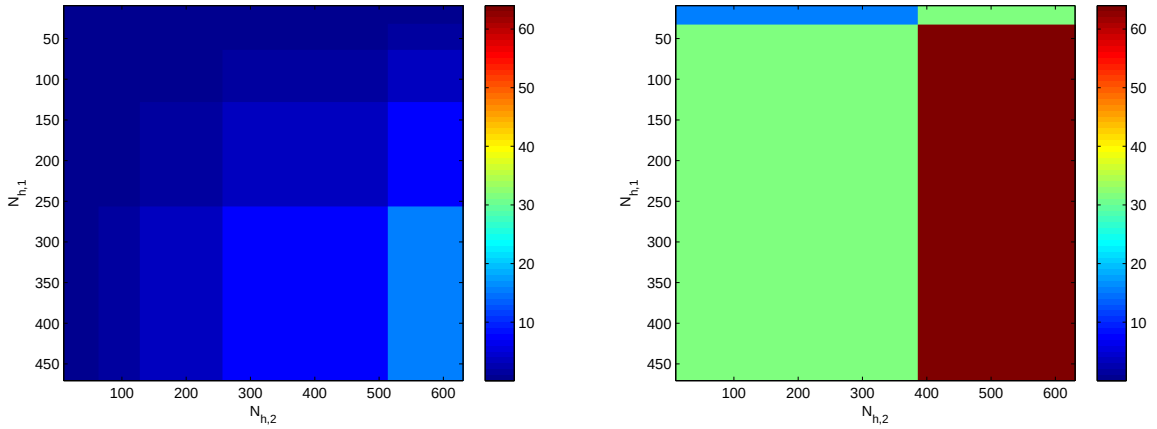
Assuming $N_d \ll N_{c,d}$, OLA and OLS offer a significant memory advantage over the conventional approach, especially for higher-dimensional data. OLA and OLS also allow the signal to be processed gradually as it is collected (e.g., for real-time signal analysis), which is more practical for a memory perspective. OLS, however, offers a distinct memory advantage over OLA. This is because OLA requires that a portion of a section's correlation output to be added to subsequent correlation outputs. In our analysis, we divide the required memory into two groups:

1. Correlation memory (CM)
2. Storage memory (SM)

CM represents the memory needed for performing correlations for each signal section. SM is the memory required to save correlation values in memory which must be added to subsequent correlation outputs which have yet to be computed. Note that OLS only requires CM, whereas OLA requires CM and SM.

For 1D, both OLA and OLS require $4N_1$ elements for each frequency domain multiplication (CM). However, the OLA algorithm must also store $N_{h,1} - 1$ (real) elements (SM) from the previous correlation result to add to the output of the current correlation.

For 2D, both OLA and OLS require $4N_1N_2$ memory elements for the CM, which comprises the total required memory. In Fig. 5.16, we compare the CM for OLA and OLS to the conventional scheme's memory requirement. Here, the image size is 480×640 and we sweep the template sizes from 10×10 to 470×630 pixels. Clearly, the CM required for OLA and OLS is



(a) CM for OLA/OLS

(b) Total Memory, Conventional

Figure 5.16: Correlation memory for OLA/OLS compared to memory required for the conventional scheme. The image size here is 480×640 and units are MB.

considerably smaller than the conventional scheme, especially for smaller template sizes. This illustrates the tremendous advantage OLA and OLS offer over the conventional scheme.

For OLA, determining the exact SM in 2D is significantly more complicated. A closed-form expression for SM does not generalize well, and is non-trivial to determine. We use indices k_1 and k_2 to denote the section index (row and column, respectively). These indices vary from 0 to $K_1 - 1$ and $K_2 - 1$, respectively (for notation simplicity, we drop the OLA subscript). The input image is divided into a $K_1 \times K_2$ grid of non-overlapping image segments measuring $L_1 \times L_2$. Note that the last elements in each row or column may be smaller. As shorthand, we will refer to the image segment at row k_1 and column k_2 as the pair (k_1, k_2).

We have two logical choices on how to process the input image: we may scan in sections left to right (row-by-row) or we may scan in sections top to bottom (column-by-column). In both cases, the instantaneous SM is a function of time. To determine what this function is, it is helpful to illustrate the regions of values that are computed for each iteration. We show the applicable regions for one 2D correlation operation for a single image segment in Fig. 5.17. In this figure, the correlation output is of size $N_1 \times N_2$. The original input segment's location coincides with region 1, which is also $L_1 \times L_2$. The values in this region are added to any overlapping regions

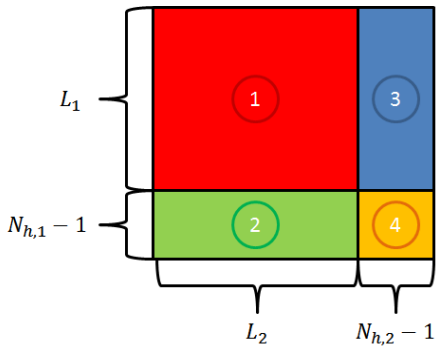


Figure 5.17: Regions of memory for one output section correlation.

from previous iterations (if applicable), and then may be processed and discarded. For example, for CFs, this region would be searched for peaks, which would constitute detections. The other three regions (2, 3, and 4) generated by the section correlation have areas (equivalent to number of elements) given by

$$A_2 = (N_{h,1} - 1)L_2 \tag{5.34}$$

$$A_3 = (N_{h,2} - 1)L_1 \tag{5.35}$$

$$A_4 = (N_{h,1} - 1)(N_{h,2} - 1) \tag{5.36}$$

Values in these regions must be stored for future iterations. Therefore, the instantaneous expression for SM will be a function of A_2, A_3, and A_4. As we process the image, some of these stored values are discarded (because they have been added to a subsequent iteration's output correlation, and are no longer needed) or are retained because they will still overlap subsequent iterations' output correlations.

We now outline the method by which we determine the instantaneous SM for the first method (row-by-row scanning). Our convention is to denote the SM needed for subsequent algorithm iterations. For example, the instantaneous SM for section iteration $(0, 0)$ is the number of elements that must be stored in memory *after* computing the first section correlation. Note that for every

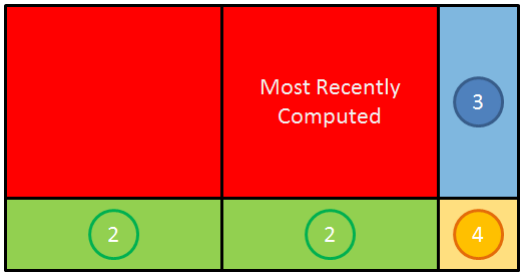


Figure 5.18: The required storage memory after the second iteration.

iteration, the elements in region 1 may be processed and then discarded. They are never needed for subsequent iterations, as there is never any overlap in these regions in subsequent iterations. However, these regions will still be shown in the following figures for clarity.

After the first section iteration $(0, 0)$, we must store $A_2 + A_3 + A_4$ elements in memory (see Fig. 5.17). The next section iteration $(0, 1)$ processes the section immediately adjacent to the first section, as shown in Fig. 5.18. In this case, we need to store $2A_2 + A_3 + A_4$ elements in memory. Following from this, the section iteration for $(0, k_2)$ (assuming $k_2 \neq K_2 - 1$), requires $(k_2 + 1)A_2 + A_3 + A_4$ elements.

The memory required for the last section in the first row is identical to the memory needed for the last section in *any* row, provide that the row is not the last row in the image. Formally, for any section iteration $(k_1, K_2 - 1)$ where $k_1 \neq K_1 - 1$, we must store $K_2A_2 + A_4$ memory elements (see Fig. 5.19). Notice that we do not need to store the elements from region 3, as they will not overlap any subsequent iterations. As a special case, if $K_1 = 1$ (i.e. the image has only one row of sections), then the memory required for processing any section iteration in the first row is $A_3 + A_4$, except for the last section iteration, where the required memory is 0.

Next, we demonstrate the memory usage for subsequent rows. In Fig. 5.20, we show the memory required for section iteration $(1, 0)$. Comparing this to the previous iteration (Fig. 5.19), the memory that was used to store region 2 from the section iteration $(0, 0)$ may now be used to store region 2 from the section iteration $(1, 0)$. However, more memory must be used to store the values directly to the right of section $(1, 0)$. From Fig. 5.20, note that region 3 from section

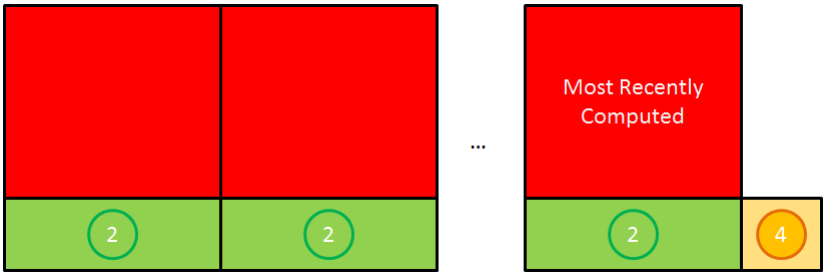


Figure 5.19: The required storage memory after the last section in any row except the last row.

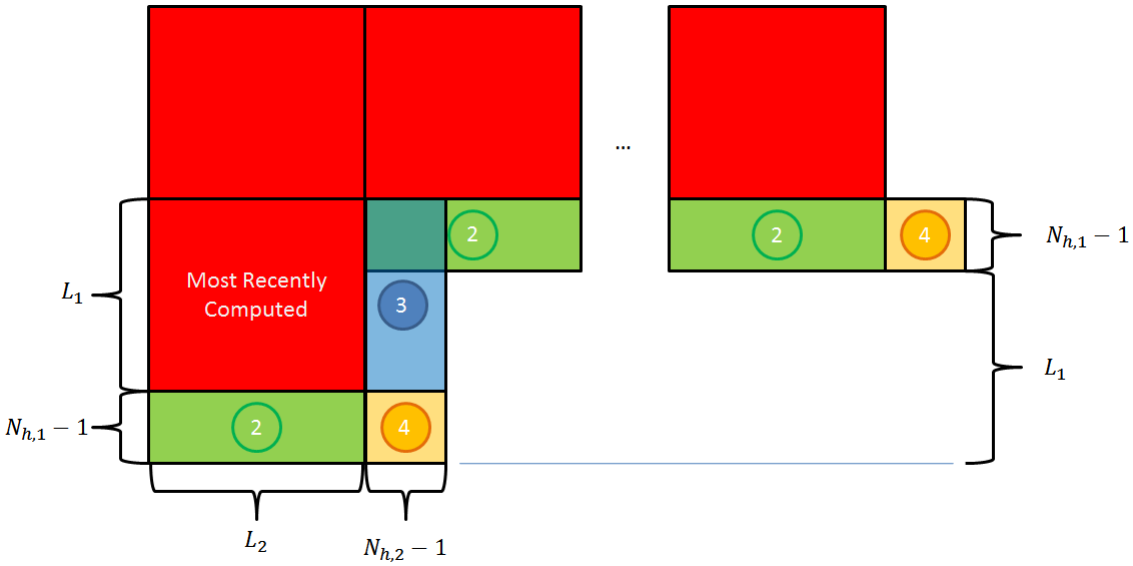


Figure 5.20: The required storage memory after the first section in the second row.

$(1,0)$ overlaps partially with region 2 from section $(0,1)$. In fact, this overlap can vary greatly in appearance depending on the dimensions of regions 2 and 3. However, due to the geometry, the *non-overlapping* region (part of region 3 and all of region 4) from section $(1,0)$ will always measure $L_1 \times (N_{h,2}-1) = A_3$. Therefore, section iteration $(1,0)$ requires $K_2A_2 + A_3 + A_4$ memory elements. In fact, this expression describes the memory requirement for every iteration for the rest of the image, except the final section in a row (as described previously) or any section in the final row (to be described next).

For the final row (for $K_1 > 1$), section iteration (K_1-1, k_2) once again generates a section of non-overlapping memory equivalent in size to A_3 that must be stored for subsequent iterations. However, region 2 for these sections does not need to be saved, as there are no more rows to be

processed. Therefore, the instantaneous SM for section $(K_1 - 1, k_2)$ is given by $(K_2 - (k_2 + 1))A_2 + A_3 + A_4$ for when $k_2 \neq K_2 - 1$. The final case is for section $(K_1 - 1, K_2 - 1)$, which requires no storage memory (as it is the last iteration).

We combine these expressions into a single expression for the instantaneous SM, given by

$$\text{SM}^{r\times r}(k_1,k_2) = \begin{cases} 0 & k_1 = K_1 - 1, k_2 = K_2 - 1 \\ \left[K_2(1-\delta(k_1)) + (k_2+1)(\delta(k_1) - I_1)\right] A_2 & else \\ \qquad +(1-I_2)A_3 + A_4 & \end{cases} \tag{5.37}$$

where

$$I_1 = \delta(k_1 - K_1 + 1) \tag{5.38}$$

$$I_2 = \delta(k_2 - K_2 + 1) \tag{5.39}$$

and $\delta()$ represents the delta function. Note that I_1 is equal to 1 for sections in the last row ($k_1 = K_1 - 1$) and 0 otherwise, and that I_2 is equal to 1 for sections in the last column ($k_2 = K_2 - 1$) and 0 otherwise. In our notation, $r \times r$ denotes that we are processing the image row-by-row. Later, we will denote column-by-column processing as $c \times c$.

Assuming[3] $K_1 > 2$ and $K_2 > 2$, the maximum SM occurs after computing a correlation after the first row is processed ($k_1 > 0$, $k_2 \geq 0$) for any $k_1 \neq K_1 - 1$ and $k_2 \neq K_2 - 1$. (Recall that an example of this situation is shown in Fig. 5.20). The stated conditions simply exclude the iteration corresponding to the final row and/or column, which require less memory because the correlation values at these edges will not overlap with any subsequent correlations.

[3]For cases where K_1 or K_2 are less than 2, there are different expressions for the maximum memory required. There are six cases: $K_1 = K_2 = 1$; $K_1 = K_2 = 2$; $K_1 = 1$, $K_2 \geq 2$; $K_1 \geq 2$, $K_2 = 1$; $K_1 = 2$, $K_2 > 2$; and $K_1 > 2$, $K_2 = 2$. In the analysis presented here, we take these six cases into account. However, we do not present the equations for each of these six cases for brevity. In most cases in practice, K_1 and K_2 will be larger than 2.

The maximum instantaneous SM is given by

$$\begin{aligned} max(\text{SM}^{r\times r}) &= K_2A_2 + A_3 + A_4 \\ &= K_2(N_{h,1}-1)L_2 + (N_{h,2}-1)L_1 + (N_{h,1}-1)(N_{h,2}-1) \end{aligned} \tag{5.40}$$

If we process the image column-by-column instead, we can go through a similar procedure to determine the instantaneous storage memory, which is given by

$$\text{SM}^{c\times c}(k_1,k_2) = \begin{cases} 0 & k_1 = K_1-1, k_2 = K_2-1 \\ \left[K_1(1-\delta(k_2)) + (k_1+1)(\delta(k_2)-I_2)\right]A_3 & else \\ \qquad +(1-I_2)A_2 + A_4 & \end{cases} \tag{5.41}$$

Assuming $K_1 > 2$ and $K_2 > 2$, the maximum memory usage occurs after computing a correlation after the first column is processed ($k_2 > 0$, $k_1 \geq 0$) for any $k_1 \neq K_1 - 1$ and $k_2 \neq K_2 - 1$. Again, these conditions simply exclude the iteration corresponding to the final row and/or column. In this case, the maximum storage memory is given by

$$\begin{aligned} max(\text{SM}^{c\times c}) &= A_2 + K_1A_3 + A_4 \\ &= (N_{h,1}-1)L_2 + K_1(N_{h,2}-1)L_1 + (N_{h,1}-1)(N_{h,2}-1) \end{aligned} \tag{5.42}$$

Note that the maximum total memory (TM) required for OLS is given by

$$\text{TM}_{OLS}^{2D} = 4N_1N_2 \tag{5.43}$$

and the maximum TM required for OLA is given by

$$\text{TM}_{OLA}^{2D} = 4N_1N_2 + max(\text{SM}^{r\times r}) \tag{5.44}$$

or

$$\text{TM}_{OLA}^{2D} = 4N_1N_2 + max(\text{SM}^{c\times c}) \tag{5.45}$$

depending on whether row-by-row or column-by-column scanning is used.

5.6.2.1 Row-by-Row vs. Column-by-Column Scanning

To determine whether to use the row-by-row or column-by-column scanning, we can compare an inequality of Eq. 5.40 and 5.42. We use the row-by-row scanning when

$$max(\text{SM}^{r\times r}) < max(\text{SM}^{c\times c})$$

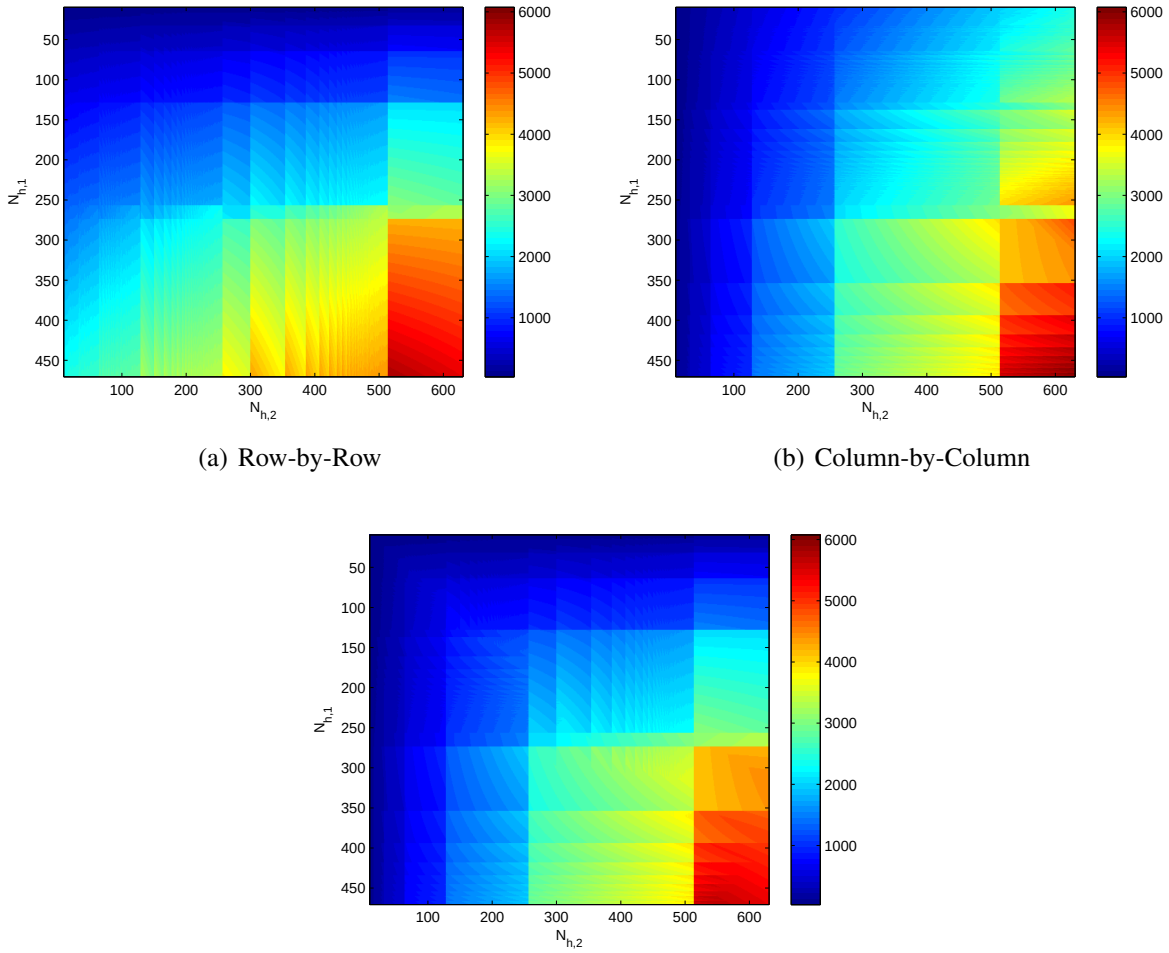
$$K_2A_2 + A_3 + A_4 < A_2 + K_1A_3 + A_4 \tag{5.46}$$

$$(K_2 - 1)A_2 < (K_1 - 1)A_3$$

$$(K_2 - 1)(N_{h,1} - 1)L_2 < (K_1 - 1)(N_{h,2} - 1)L_1 \tag{5.47}$$

and the column-by-column scanning otherwise. If both sides of the inequality in 5.47 are equal, then either scanning may be used. In the simplest of cases, if $N_{h,1} = N_{h,2}$ and $L_1 = L_2$, then row-by-row scanning should be used when $K_2 < K_1$. By choosing the scanning carefully, memory usage may be reduced.

To illustrate this, we compute the required memory for OLS and compare it to OLA using both the row-by-row and column-by-column scanning methods. In addition, we show results for a dynamic scheme which will choose row-by-row or column-by-column scanning based on the OLA parameters. We assume a test image of size $N_{m,1} \times N_{m,2}$ and sweep the template sizes from 10×10 to $(N_{m,1} - 10) \times (N_{m,2} - 10)$.



(a) Row-by-Row

(b) Column-by-Column

(c) Dynamic

Figure 5.21: Required SM, shown in kB, for the three OLA schemes, using $q_d = 1$. The image size is 480×640.

We use an FFT size according to Eq. 5.5, with $q_d = 1$. We compute the additional memory required by OLA to store correlation values for subsequent iterations (Eq. 5.40 and 5.42). In Fig. 5.21, we show the maximum SM for each scheme for a test image of size 480×640. Notice that as the template size gets larger, the memory requirements in general get larger. In both cases, the dynamic scheme improves memory use by selecting the best scanning direction.

To see this more clearly, we compare OLA to OLS for this case. We compute the ratio of the OLA SM to the OLS TM (TM^{2D}_{OLS}). Recall that TM^{2D}_{OLS} is the same as the CM required by both algorithms. For example, a factor of 1 means that OLA requires 100% more memory

Table 5.3: Largest Ratio of SM to CM ($q_d = 1$.)

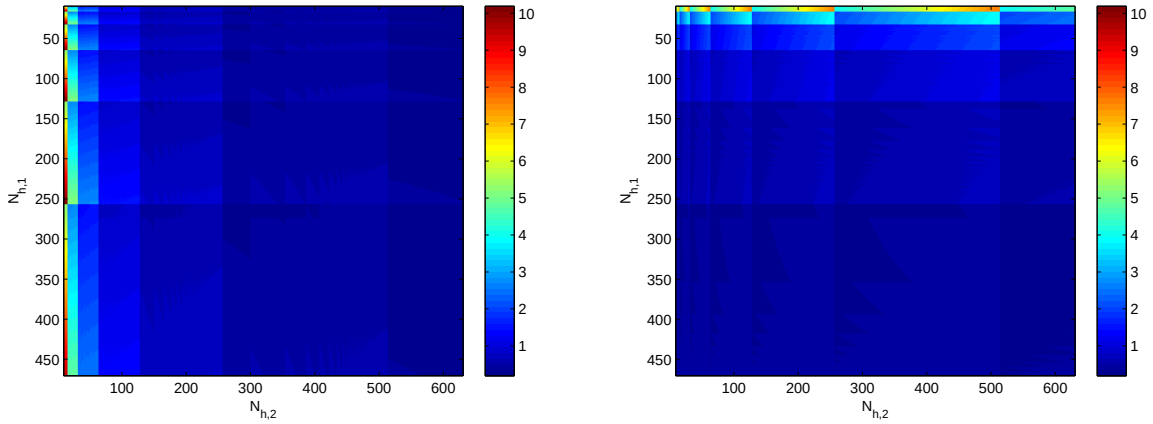
Resolution	Row-by-Row	Column-by-Column	Dynamic
240×320	$N_h = 128 \times 16$ OLS: 64 kB, OLA: 396.5 kB (5.2)	$N_h = 16 \times 256$ OLS: 128 kB, OLA: 636.1 kB (4.0)	$N_h = 16 \times 16$ OLS: 8 kB, OLA: 38 kB (3.8)
480×640	$N_h = 256 \times 16$ OLS: 128 kB, OLA: 1433 kB (10.2)	$N_h = 16 \times 512$ OLS: 256 kB, OLA: 2232.3 kB (7.7)	$N_h = 16 \times 16$ OLS: 8 kB, OLA: 66.1 kB (7.3)
480×720	$N_h = 256 \times 16$ OLS: 128 kB, OLA: 1592.4 kB (11.4)	$N_h = 16 \times 512$ OLS: 256 kB, OLA: 2232.3 kB (7.7)	$N_h = 16 \times 16$ OLS: 8 kB, OLA: 66.1 kB (7.3)
576×720	$N_h = 512 \times 16$ OLS: 256 kB, OLA: 3190.4 kB (11.5)	$N_h = 16 \times 512$ OLS: 256 kB, OLA: 2615.5 kB (9.2)	$N_h = 16 \times 16$ OLS: 8 kB, OLA: 77.4 kB (8.7)
600×800	$N_h = 512 \times 16$ OLS: 256 kB, OLA: 3509.8 kB (12.7)	$N_h = 16 \times 512$ OLS: 256 kB, OLA: 2711.3 kB (9.6)	$N_h = 16 \times 16$ OLS: 8 kB, OLA: 80.2 kB (9.0)
720×1280	$N_h = 512 \times 16$ OLS: 256 kB, OLA: 5426 kB (20.2)	$N_h = 16 \times 1024$ OLS: 512 kB, OLA: 6386.4 kB (11.5)	$N_h = 16 \times 16$ OLS: 8 kB, OLA: 94.3 kB (10.8)
1080×1920	$N_h = 1024 \times 16$ OLS: 512 kB, OLA: 15977 kB (30.2)	$N_h = 16 \times 1024$ OLS: 512 kB, OLA: 9263.6 kB (17.1)	$N_h = 16 \times 16$ OLS: 8 kB, OLA: 136.4 kB (16.1)

than OLS, and equivalently that SM is equal to CM. We show this ratio in Fig. 5.22for all three methods. In this example, OLA requires a minimum of at least ~19% more memory than OLS for each method. For the row-by-row scanning, tall and narrow templates require significantly more SM than CM; for the column-by-column case, short and wide templates require more SM than CM. The dynamic scheme results in the best performance, reducing the ratio of SM to CM significantly for many of the templates of these sizes. In Table 5.3, we show the largest ratio of SM to CM from Fig. 5.22 (in addition to several other common resolutions) and how much memory is required in these cases. Note that the highest ratio for the dynamic scheme occurs at very small template sizes, which require very little memory to begin with. Also note that the SM to CM ratio increases with test image resolution. Finally, note that, in all cases, the highest SM to CM ratio is the lowest for the dynamic scheme.

It is clear that OLA requires more memory than OLS. However, this drawback can be minimized by paying careful attention to how sections are scanned while processing an input signal.

5.6.2.2 Effect of FFT Size on Memory

Next, we investigate the choice of N_d on memory requirements. A larger N_d will require more CM and SM. For the conventional scheme, there is no benefit to using $q_d > 1$, as it only hurts



(a) Row-by-Row

(b) Column-by-Column

(c) Dynamic

Figure 5.22: Factor by which OLA storage memory exceeds OLS memory for the three OLA schemes, using $q_d = 1$. The image size is 480×640.

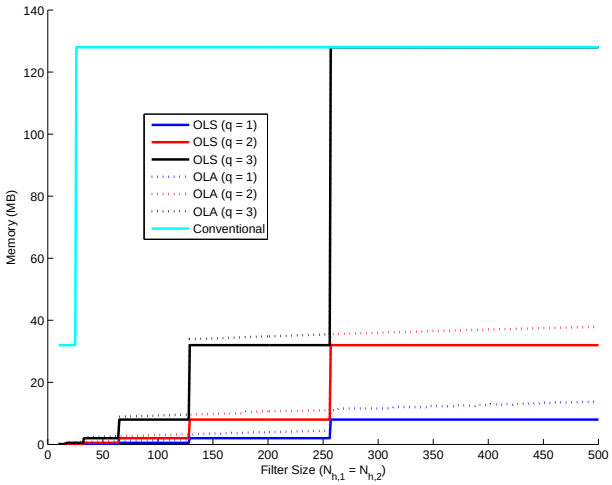


Figure 5.23: The effects of varying q_d and the template size. Here, the template is always a square ($N_{h,1} = N_{h,2}$) and the image size is 1000×1000 pixels.

computational performance and increases memory. On the other hand, we demonstrated in Section 5.5 that OLA and OLS often perform better for larger values of q_d. There is a corresponding memory tradeoff that comes with this improved computation. Choosing a larger value for q_d will increase FFT size and subsequently increase both CM and SM. In this section, we quantify this.

We illustrate the memory requirements for all three methods in Fig. 5.23. To simplify the discussion, we restrict $N_{h,1} = N_{h,2}$ and $N_{m,1} = N_{m,2}$. Note that this makes the row-by-row and column-by-column sequencing for OLA generate identical results. We use a test image of size 1000×1000, but results are similar for differently-sized images. We sweep the template size from 10×10 to 500×500. For OLA and OLS, as the template size grows, N_1 and N_2 step up according to q_d. For $q_d = 3$ and $N_{h,d} > 256$, the OLA and OLS schemes actually converge to the conventional scheme's memory usage, as they each contain only one section each. In this case, OLA requires no storage memory. Otherwise, we note that OLA requires significantly more memory than OLS.

Finally, we consider the case where $N_1$and N_2 are fixed and equal. We maintain the image size at 1000×1000 and vary the template sizes from 10×10 to $N_1 \times N_2$ as shown in Fig. 5.24. We note that small templates require a small amount of SM. As the templates get larger, the SM

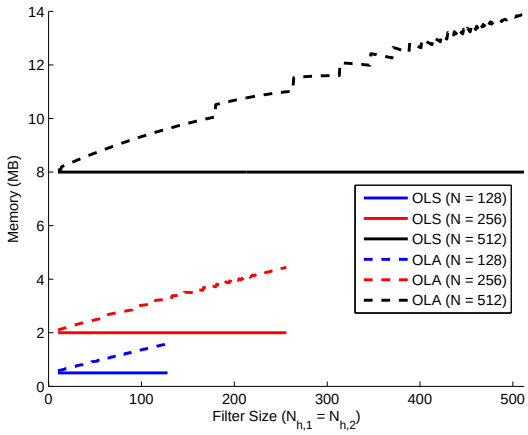


Figure 5.24: The effects of varying N_d and the template size. Here, the template is always a square ($N_{h,1} = N_{h,2}$) and the image size is 1000x1000 pixels.

increases. When $N_{h,d} = N_d$, OLA requires 3.18, 2.22, and 1.74 times as much memory as OLS for $N_d = 128$, $N_d = 256$, and $N_d = 512$, respectively.

5.7 Memory Analysis for 3D Applications

We now discuss memory requirements in 3D. We use a video of size $N_{m,1} \times N_{m,2} \times N_{m,3}$ that is to be processed by a template of size $N_{h,1} \times N_{h,2} \times N_{h,3}$. We consider two different methods by which we can do this processing. In the first method, we perform OLA and OLS in all three dimensions. In the second method, we perform OLA and OLS in only one dimension (the temporal dimension, $d = 3$). This method is easier to implement.

Before beginning, we note that the conventional scheme features an instantaneous memory of $4N_{c,1}N_{c,2}N_{c,3}$. For desktop computers, this memory often exceeds the amount of available RAM for even low resolutions videos and a few hundred frames. Therefore, OLA and OLS are absolutely essential for processing videos.

5.7.1 OLA and OLS in All Dimensions

First, we perform OLA and OLS in all three dimensions. We make some assumptions based on the domain of the application. First, we assume that a group of frames must be read into memory. We denote this group of frames as a *video slice*. To simplify our analysis, we divide the required memory into four groups:

1. Video Slice Memory (VSM)
2. Correlation Memory (CM)
3. Current Slice Storage Memory (CSSM)
4. Future Slice Storage Memory (FSSM)

OLS only requires 1 and 2, whereas OLA requires 1-4. This is because partial correlation values from each spatio-temporal segment must be stored both for adding to other correlation outputs from the same video slice (3) and future video slices (4).

For OLS, we must load in N_3frames at a time. The memory required to store these frames (VSM) is

$$\text{VSM}_{OLS} = N_{m,1}N_{m,2}N_3 \tag{5.48}$$

This video segment is then divided into $L_1 \times L_2 \times N_3$ blocks that are transformed via $N_1 \times N_2 \times N_3$ size FFTs. Therefore, the memory required for one correlation operation (CM) is

$$\text{CM}_{OLS} = 4N_1N_2N_3 = 4N^{\#} \tag{5.49}$$

For OLA, we must load in L_3 frames at a time. The memory required to store these frames (VSM) is

$$\text{VSM}_{OLA} = N_{m,1}N_{m,2}L_3 \tag{5.50}$$

Note that this serves as an advantage over OLS, which must store $\text{VSM}_{OLS} - \text{VSM}_{OLA} = N_{m,1}N_{m,2}(N_3 - L_3) = N_{m,1}N_{m,2}(N_{h,3} - 1)$ more memory elements. The video slice segment

is then divided into a grid of $K_1 \times K_2$ non-overlapping blocks of dimension $L_1 \times L_2 \times L_3$ that are transformed via $N_1 \times N_2 \times N_3$ size FFTs prior to element-wise multiplication. Note that $\text{CM}_{OLA} = \text{CM}_{OLS}$ (see Eq. 5.49).

OLA must store a portion of the outputs for each iteration in memory, so that they may be added to subsequent outputs. Determining the total storage memory is simplified by computing the CSSM and the FSSM separately. Fig. 5.25 shows one segment of a 3D correlation output. As in the 2D case, region 1 is the portion of the correlation output that coincides with the original $L_1 \times L_2 \times L_3$ input segment. Regions 2-8 are output correlation values that will be added to the outputs of subsequent correlations. The volumes of the regions are given as follows,

$$V_1 = L_1 L_2 L_3 \tag{5.51}$$

$$V_2 = (N_{h,1} - 1) L_2 L_3 \tag{5.52}$$

$$V_3 = (N_{h,2} - 1) L_1 L_3 \tag{5.53}$$

$$V_4 = (N_{h,1} - 1)(N_{h,2} - 1) L_3 \tag{5.54}$$

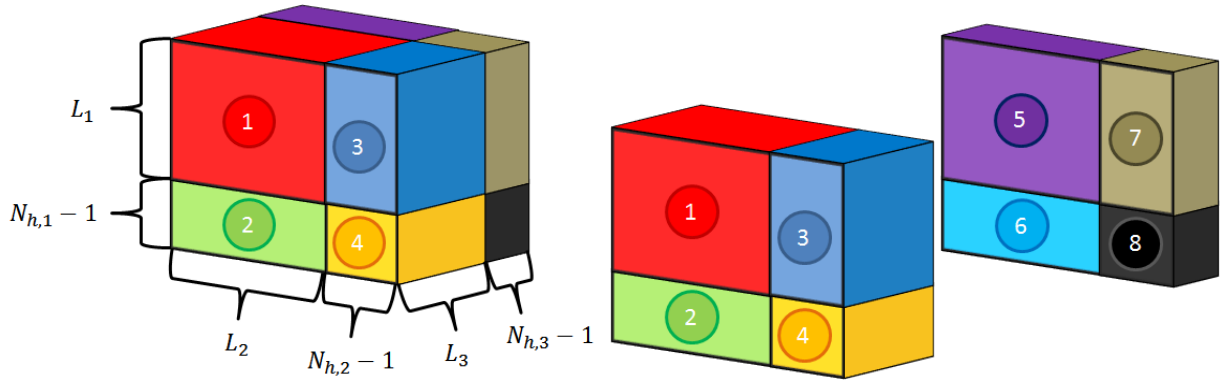
$$V_5 = L_1 L_2 (N_{h,3} - 1) \tag{5.55}$$

$$V_6 = (N_{h,1} - 1)(N_{h,3} - 1) L_2 \tag{5.56}$$

$$V_7 = (N_{h,2} - 1)(N_{h,3} - 1) L_1 \tag{5.57}$$

$$V_8 = (N_{h,1} - 1)(N_{h,2} - 1)(N_{h,3} - 1) \tag{5.58}$$

By categorizing the type of SM, it is clear that elements from regions 2, 3, and 4 contribute to the CSSM. These values consists of correlation outputs that are added to other correlation outputs generated from cuts from the *current* video slice. Correlation elements from regions 5, 6, 7, and 8 contribute to the FSSM. These values consist of correlation outputs that are added to other correlation outputs generated from cuts from one or more *future* video slices.



(a) Dimensions of Regions (b) Expanded View

Figure 5.25: Regions associated with each 3D correlation result.

Determining the CSSM is a problem that is similar to the 2D memory case, with the exception that the areas are now volumes. Recall that the indices k_1 and k_2 vary from 0 to K_1-1 and K_2-1, respectively. Assuming row-by-row scanning, the instantaneous memory required for the CSSM is, as a function of k_1 and k_2,

$$\text{CSSM}^{r\times r}(k_1,k_2)=\begin{cases}0 & k_1=K_1-1,k_2=K_2-1\\ \left[K_2(1-\delta(k_1))+(k_2+1)(\delta(k_1)-I_1)\right]V_2 & \\ \qquad +(1-I_2)V_3+V_4 & else\end{cases} \tag{5.59}$$

Note that I_1 and I_2 were defined in Eq. 5.38 and 5.39, respectively. For column-by-column scanning, we have

$$\text{CSSM}^{c\times c}(k_1,k_2)=\begin{cases}0 & k_1=K_1-1,k_2=K_2-1\\ \left[K_1(1-\delta(k_2))+(k_1+1)(\delta(k_2)-I_2)\right]V_3 & \\ \qquad +(1-I_1)V_2+V_4 & else\end{cases} \tag{5.60}$$

We next determine the FSSM. This problem is very similar to the CSSM problem. Note that the FSSM regions are analogous to the CSSM regions, with region 6 being analogous to region 2, region 7 analogous to region 3, and region 8 analogous to region 4. In this case, values in region 5 must also be stored (whereas values in region 1 were processed and discarded in the CSSM case). Also, we may not discard any values in the last row or column, as *all* values from these regions must be stored to be added to future correlation outputs. To be clear, we illustrate the process for determining FSSM next.

Once again, we assume row-by-row scanning. Recall that we refer to the video segment at row k_1 and column k_2 simply as the pair (k_1, k_2). Note that we do not include the index k_3 here as this process is the same for each video slice. Our convention is to denote the FSSM needed for subsequent algorithm iterations. For example, the instantaneous FSSM for section iteration $(0, 0)$ is the number of elements that must be stored in FSSM *after* computing the correlation for section $(0, 0)$.

After the first section iteration $(0, 0)$, the future slice storage memory must store $V_5 + V_6 + V_7 + V_8$ elements in memory (refer to Fig. 5.25). The next section iteration $(0, 1)$ process the section immediately to the right of the first section (see Fig. 5.26). After this iteration, we must store $2V_5 + 2V_6 + V_7 + V_8$ memory elements in the future slice storage memory. Generalizing, the amount of FSSM necessary for segment $(0, k_2)$ requires $k_2V_5 + k_2V_6 + V_7 + V_8$ memory elements. Note that the last section in a row (see Fig. 5.27) requires $K_2V_5 + K_2V_6 + V_7 + V_8$ memory elements.

Next, we demonstrate FSSM usage for subsequent rows. In Fig. 5.20, we show the memory required for section iteration $(1, 0)$. Comparing this to the previous iteration (Fig. 5.27), the memory that was used to store region 6 from the section iteration $(0, 0)$ may now be used to store region 6 from the section iteration $(1, 0)$. From Fig. 5.28, note that region 7 from section $(1, 0)$ overlaps partially with region 6 from section $(0, 1)$. In fact, this overlap can vary greatly in appearance depending on the dimensions of regions 6 and 7. However, due to the geometry, the *non-overlapping* region (part of region 7 and all of region 8) from section $(1, 0)$ will always

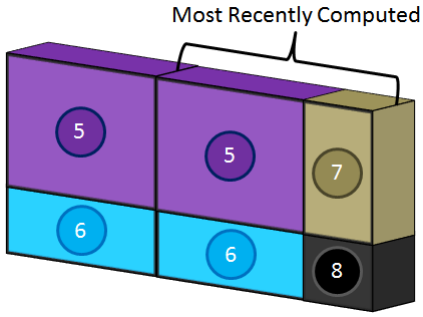


Figure 5.26: The required FSSM in the first row after the second iteration.

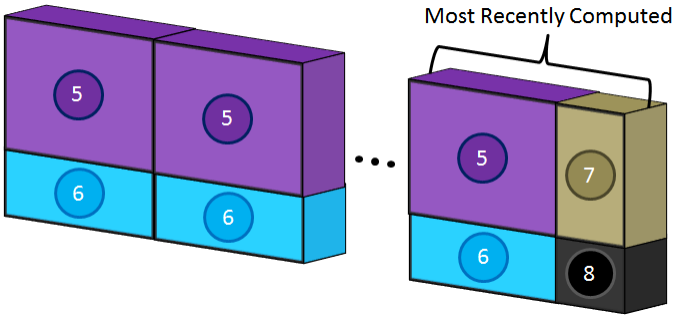


Figure 5.27: The required FSSM for the section in the first row and last column.

measure $L_1 \times (N_{h,2} - 1) \times (N_{h,3} - 1) = V_7$. Therefore, section iteration $(1, 0)$ requires $(K_2 + 1)V_5 + K_2V_6 + 2V_7 + V_8$ memory elements. This expression is similar for the remaining sections in the second row, with the exception that the V_5 coefficient will always grow as new sections are added to the FSSM.

Unlike the CSSM, the FSSM does not exhibit any special cases for the final row of sections. This is because every value in regions 5-8 must be stored to be added to future outputs in the next video slice. We combine the above expressions into a single expression for the instantaneous memory, given by

$$\text{FSSM}^{r\times r}(k_1,k_2,1) = (k_1K_2+k_2+1)V_5+[K_2+(k_2+1-K_2)\delta(k_1)]\,V_6+(k_1+1)V_7+V_8 \quad (5.61)$$

For column-by-column scanning, FSSM is given by

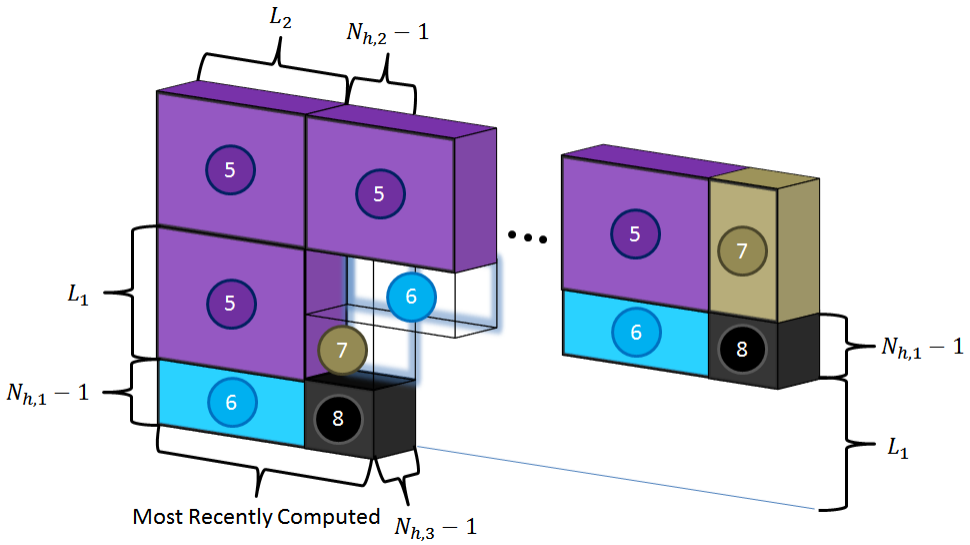


Figure 5.28: The required storage memory after the first section in the second row.

$$\text{FSSM}^{c\times c}(k_1,k_2,1) = (k_2K_1+k_1+1)V_5+(k_2+1)V_6+[K_1+(k_1+1-K_1)\delta(k_2)]\,V_7+V_8 \quad (5.62)$$

Note that the above expression only holds true for the first video slice. In this case, the FSSM increases monotonically as we process the slice, reaching a maximum after the entire video slice has been processed. For subsequent slices, the FSSM is constant, and equal to the maximum (and final) value $\text{FSSM}^{r\times r}(K_1-1,K_2-1,1) = \text{FSSM}^{c\times c}(K_1-1,K_2-1,1) = K_1K_2V_5+K_2V_6+K_1V_7+V_8$. This is because the FSSM stores correlation values from the slice at k_3-1 when the slice at $k_3>0$ is being processed. This memory is gradually replaced by new correlation values generated from the slice at k_3 that are stored for processing the slice at k_3+1.

Unlike the FSSM, the CSSM increases (but tends to stay approximately constant) while processing *any* video slice. The CSSM periodically decreases at the end of each row and goes to zero after the last row is processed. To illustrate these trends, we show an example of the different memory components in Fig. 5.29. In this example, we have a video of size $N_m = 1080\times 1920\times 1000$ that is correlated with a template of size $N_h = 50\times 50\times 50$. We

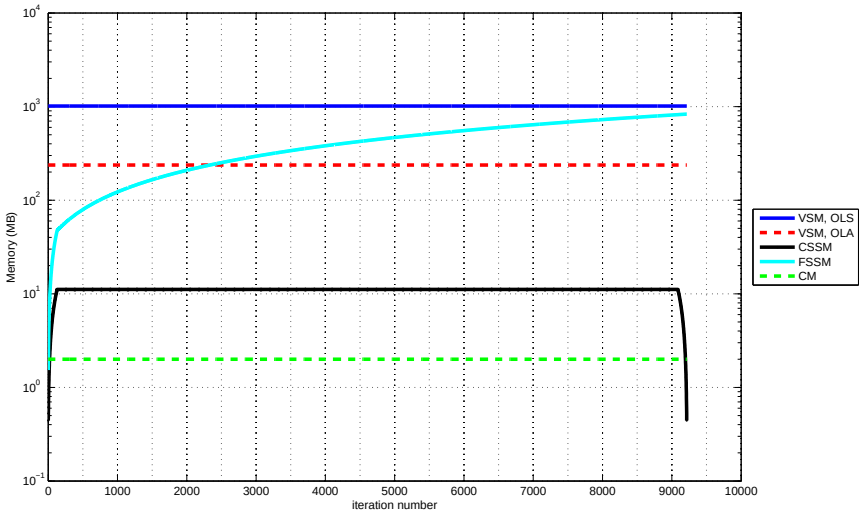


Figure 5.29: The memory components involved in OLA and OLS correlations for a video of size $N_m = 1080 \times 1920 \times 1000$ with a template of size $N_h = 50 \times 50 \times 50$. Note that this is the first video slice, and as a result the FSSM starts at zero and increases as the slice is processed.

have used $q_d = 1$ in all dimensions, which affects CM, CSSM, and FSSM. To illustrate the effect of q_d, we repeat this situation for values of 1, 2, and 3 for q_d and show each type of memory in Fig. 5.30.

The instantaneous TM required to process the video using OLS is given by

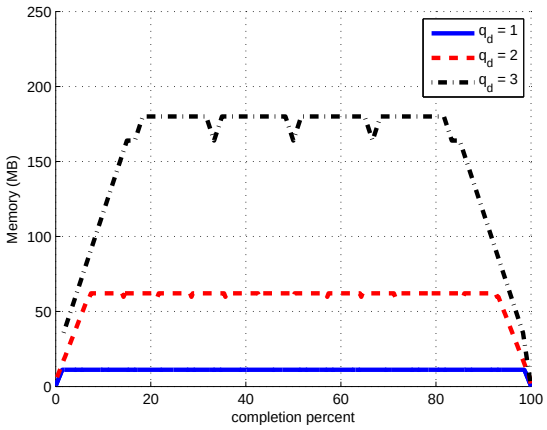
$$\text{TM}_{OLS}^{3D} = \text{VSM}_{OLS} + \text{CM}_{OLS} \tag{5.63}$$

and the instantaneous TM required to process the video using OLA is given by

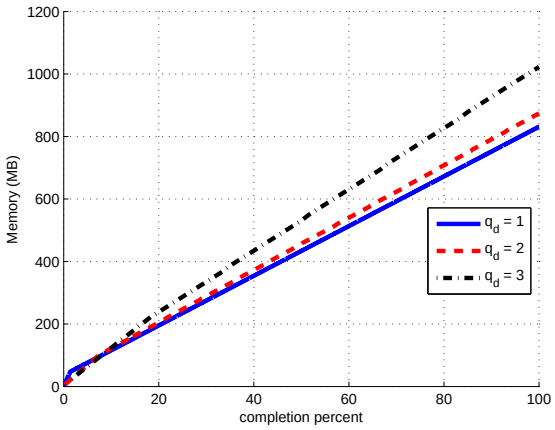
$$\text{TM}_{OLA}^{3D} = \text{VSM}_{OLA} + \text{CM}_{OLA} + \text{CSSM}^{r\times r} + \text{FSSM}^{r\times r} \tag{5.64}$$

for row-by-row processing and

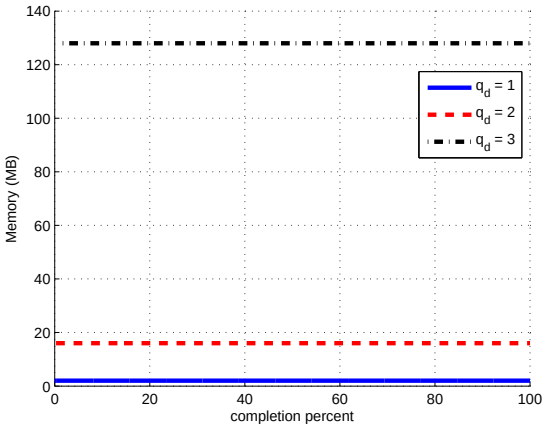
$$\text{TM}_{OLA}^{3D} = \text{VSM}_{OLA} + \text{CM}_{OLA} + \text{CSSM}^{c\times c} + \text{FSSM}^{c\times c} \tag{5.65}$$



(a) CSSM



(b) FSSM



(c) CM

Figure 5.30: The effect of q_d on CSSM, FSSM, and CM. Note that the x-axis has been normalized because the number of iterations is different from case to case (less iterations with increasing q_d). Also note that the FSSM represents the behavior for the first video slice; after this, the FSSM is constant.

for column-by-column processing. As was pointed out previously, the VSM for OLS is greater than the VSM for OLA, i.e. $\text{VSM}_{OLS} - \text{VSM}_{OLA} = N_{m,1}N_{m,2}(N_3 - L_3) = N_{m,1}N_{m,2}(N_{h,3} - 1)$. Therefore, we must ask ourselves if there is a situation in which the OLA TM is less than the OLS TM.

To investigate this, we determine the maximum memory required for OLA (note that $max(\text{TM}_{OLS}^{3D}) = \text{TM}_{OLS}^{3D}$ because the OLS memory does not change as a function of iteration number). Note that the maximum OLA CSSM (for row-by-row scanning) is given by

$$max(\text{CSSM}^{r\times r}) = K_2V_2 + V_3 + V_4 \tag{5.66}$$

which is similar to the 2D case found in Eq. 5.40. Next, the maximum OLA FSSM is given by

$$max(\text{FSSM}^{r\times r}) = \text{FSSM}^{r\times r}(K_1 - 1, K_2 - 1) = K_1K_2V_5 + K_2V_6 + K_1V_7 + V_8 \tag{5.67}$$

The maximum OLA TM is given by

$$max(\text{TM}_{OLA}^{3D}) = \text{VSM}_{OLA} + \text{CM}_{OLA} + max(\text{CSSM}^{r\times r}) + max(\text{FSSM}^{r\times r}) \tag{5.68}$$

If OLA were to be better than OLS from a memory perspective, the following condition would hold

$$max(\text{TM}^{3D}_{OLA}) < max(\text{TM}^{3D}_{OLS})$$

$$\text{VSM}_{OLA} + \text{CM}_{OLA} + max(\text{CSSM}^{r\times r}) + max(\text{FSSM}^{r\times r}) < \text{VSM}_{OLS} + \text{CM}_{OLS}$$

$$\text{VSM}_{OLS} - \text{VSM}_{OLA} > max(\text{FSSM}^{r\times r}) + max(\text{CSSM}^{r\times r})$$

$$N_{m,1}N_{m,2}(N_{h,3} - 1) > max(\text{FSSM}^{r\times r}) + max(\text{CSSM}^{r\times r}) \tag{5.69}$$

Let us examine just the first term of the right hand side. Note that the maximum FSSM memory describes the entire correlation output that is stored for a future slice. Rather than manipulate Eq. 5.67, we can simply look at the situation graphically in Fig. 5.31. Note that the maximum FSSM memory may also be written as

$$max(\text{FSSM}^{r\times r}) = [(K_1 - 1)L_1 + N_1]\,[(K_2 - 1)L_2 + N_2]\,[N_{h,3} - 1] \tag{5.70}$$

Note that the following always holds true:

$$(K_1 - 1)L_1 + N_1 \geq N_{m,1} \tag{5.71}$$

$$(K_2 - 1)L_2 + N_2 \geq N_{m,2} \tag{5.72}$$

The equality holds true for Eq. 5.71 when $N_{h,1} = 1$ and for Eq. 5.72 when $N_{h,2} = 1$. Thus $max(\text{FSSM}^{r\times r})$ is in most cases larger than the left hand side of Eq. 5.69. This is without even including the term CSSM, which is always positive. Thus the condition in Eq. 5.69 is never true, which means that, despite having a larger VSM, the maximum TM for OLS will always be less than the maximum TM for OLA. This analysis also applies to column-by-column scanning.

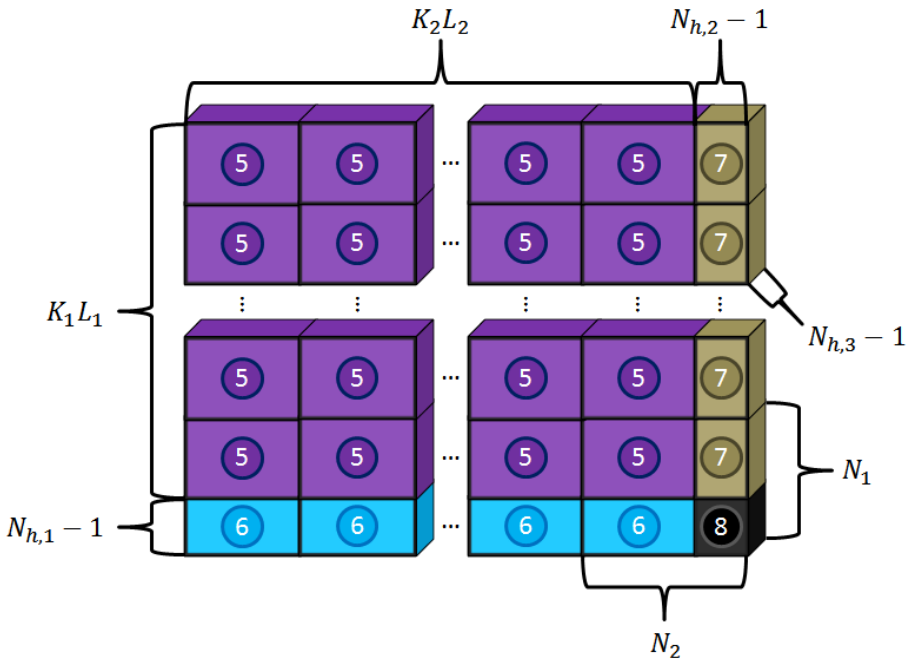


Figure 5.31: The maximum FSSM for OLA.

5.7.1.1 Effect of FFT Size on Memory

Next, we investigate the choice of N_d on memory requirements. A larger N_d will require more CM, CSSM, and FSSM. In Fig. 5.32, we plot the effect of q_d and template size on memory. To visualize the memory requirements more easily, we restrict our discussion to only square shaped template and videos ($N_{h,1} = N_{h,2} = N_{h,3}$ and $N_{m,1} = N_{m,2} = N_{m,3}$). We use a video measuring $1000 \times 1000 \times 1000$, although the temporal dimension does not matter as the video is processed in slices. Results are similar for differently-sized videos. We sweep the template size from $10 \times 10 \times 10$ to $500 \times 500 \times 500$. As the template size grows, N_1, N_2, and N_3 step up accordingly by powers of two. Note that for $N_{h,d} > 25$, the conventional scheme jumps to 256 GB. When $q_d = 3$ and $N_{h,d} > 256$, the OLA and OLS schemes converge to the conventional scheme's memory usage (256 GB), as they each contain only one section each. In this case, VSM is zero, and OLA requires no storage memory. This is not shown on the plot to allow for detail of the other regions. We observe that OLA requires significantly more memory than OLS. This detail should not be overlooked for memory-critical applications.

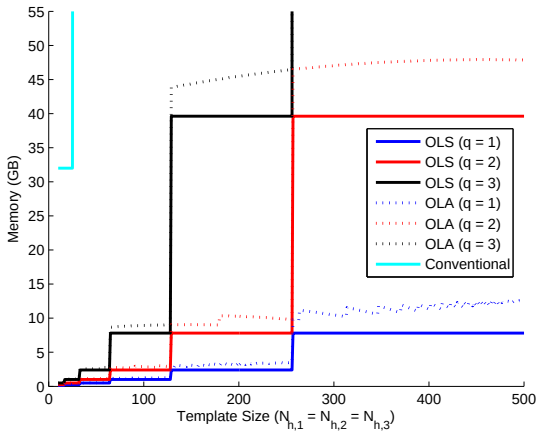
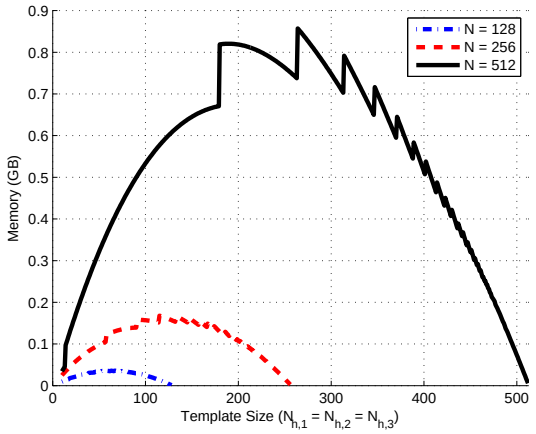
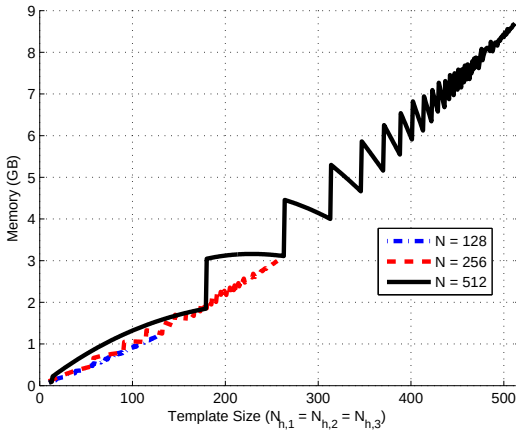


Figure 5.32: The effects of varying q_d and the template size. Here, the template is always a square ($N_{h,1} = N_{h,2} = N_{h,3}$) and the video size is $1000 \times 1000 \times 1000$ pixels.

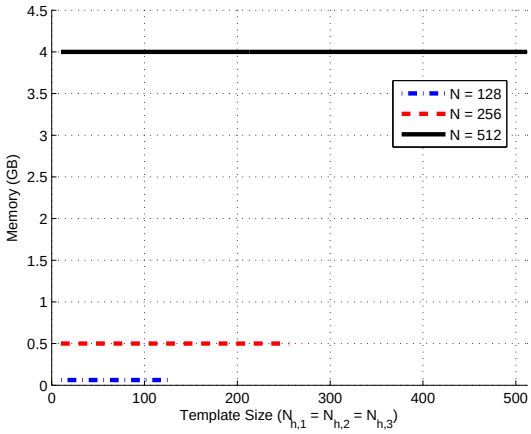
Finally, we consider the case where N_1, N_2, and N_3 are fixed and equal. We maintain the video size at $1000 \times 1000 \times 1000$. In this case, we vary the template sizes from $10 \times 10 \times 10$ to $N_1 \times N_2 \times N_3$. For small values of $N_{h,d}$, L_d is close in size to N_d. As $N_{h,d}$ approaches N_d, L_d approaches 1. This means that the number of sections, K_d, approaches the signal length, $N_{m,d}$. We have plotted $max(\text{CSSM}^{r\times r})$, $max(\text{FSSM}^{r\times r})$, and CM_{OLA} in Fig. 5.33 as a function of template size. Note that because the signals are equal in size in each dimension, column-by-column scanning is identical to row-by-row scanning. Recall that the maximum FSSM has a spatial area given by $[(K_1 - 1)L_1 + N_1]\,[(K_2 - 1)L_2 + N_2]$. Note that the terms $(K_d - 1)L_d$ will oscillate, which causes the FSSM to also oscillate. For example, for $N_d = 512$, and $N_{h,d} = 179$, the section size will be $L_d = 334$ and the number of sections will be $K_d = \lceil N_{m,d}/L_d \rceil = 3$. For a template just one sample larger, $N_{h,d} = 180$, the section size will be $L_d = 333$, which will cause $K_d = 4$. In this case, we have 3 sections that have 333 samples (in each dimension) and one section that has 1 sample. This causes the total size of the FSSM to spike from 1.85 GB to 3.05 GB. Therefore, this represents a very poor choice in parameters. Whenever the number of sections increases, the result is a spike in the FSSM. We notice a similar spike in memory for the CSSM. In this case, notice the term $K_2V_2 = K_2(N_{h,1} - 1)L_2L_3$ from Eq. 5.66. Since $L_2 = L_3$ in this example, the effective term $K_dL_d^2$ will still spike when K_d steps up. However, the product L_d^2



(a) CSSM



(b) FSSM



(c) CM

Figure 5.33: The effect of varying N_d and $N_{h,d}$ on CSSM, FSSM, and CM.

will approach 1 as $N_{h,d}$ approaches N_d. At the same time, K_d will increase to $N_{m,d}$. Therefore, the entire term will be equal to $N_{m,d}$ when $N_{h,d} = N_d$. This decrease is made evident in the behavior of the CSSM for this example, which is shown in Fig. 5.33.

In Fig. 5.34, we show the effects of $N_{h,d}$ and N_d on TM_{OLA}^{3D} and TM_{OLS}^{3D}. In general, as the templates get larger, the storage memory increases with some oscillation (due to the CSSM and FSSM effects described above). We also notice that as N_d increases, the OLA to OLS memory ratio increases. This is in contrast to the 2D case, where the opposite trend was noticed. In the 2D case, when $N_{h,d} = N_d$, OLA required 3.18, 2.22, and 1.74 times as much memory as OLS

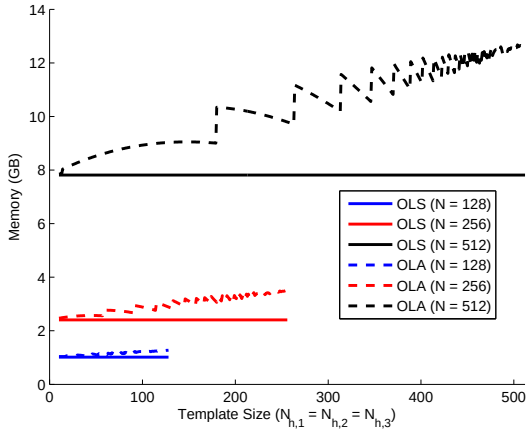


Figure 5.34: The effects of varying N_d and the template size. Here, the template is always a square ($N_{h,1} = N_{h,2} = N_{h,3}$) and the video size is $1000 \times 1000 \times 1000$ pixels. The numerical analysis here confirms that OLA always requires more memory than OLS, despite requiring less VSM.

for $N_d = 128$, $N_d = 256$, and $N_d = 512$, respectively. In the 3D case, OLA requires 1.25, 1.45, and 1.63 times as much memory as OLS for $N_d = 128$, $N_d = 256$, and $N_d = 512$, respectively.

5.7.1.2 VSM Considerations

Because the OLA scheme requires an increase in memory from the initial state (in the form of CSSM and/or FSSM), we consider an optimized implementation that would free up memory allocated to the VSM as each section is processed. This freed up memory can be used to store values for the CSSM and FSSM.

To illustrate this concept, refer to Fig. 5.35. The VSM measures $N_{m,1} \times N_{m,2} \times B$. Here, B is the temporal length of the video slice, and is given by

$$B = \begin{cases} L_3 & k_3 \neq K_3 - 1 \\ N_{m,3} - K_3 L_3 + L_3 & k_3 = K_3 - 1 \end{cases} \tag{5.73}$$

Note that the second half of this expression is the number of inclusive frames in the last video slice between the start cut length index in Table 5.1 and the last frame of the video. The VSM is divided into $L_1 \times L_2 \times B$ blocks. Spatially, these blocks form a $K_1 \times K_2$ grid. However,

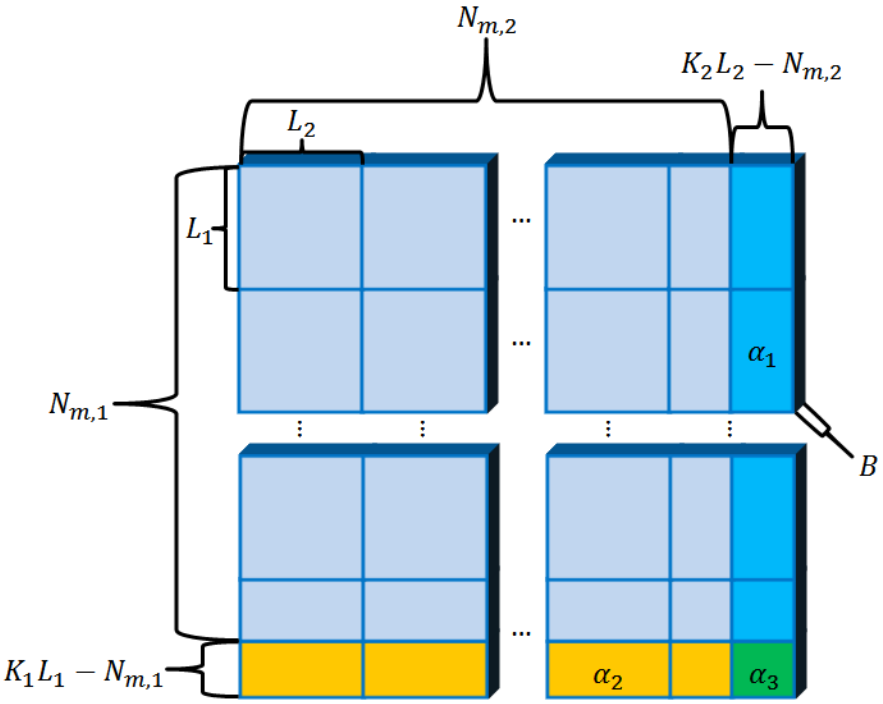


Figure 5.35: A visualization of the OLA VSM. Note that the bright blue, yellow, and green sections represent zero padding that is necessary for the sections at the end of a row or column to measure $L_1 \times L_2$.

the video dimension does not need to be a perfect multiple of the block size (i.e. $N_{m,1}$ is not necessarily a multiple of L_1, and $N_{m,2}$ is not necessarily a multiple of L_2). In these cases, there is a section of zeros appended to these blocks to make them of size $L_1 \times L_2 \times L_3$ (this is typically done along with the zero padding that makes the blocks of size $N_1 \times N_2 \times N_3$ prior to FFT). However, note that this zero padding is *not* part of the VSM, but rather is accounted for by the CM. In the figure, this zero padding is shown in the bright blue, yellow, and green regions. The bright blue regions of spatial area α_1 are appended to the right of sections in the final column ($k_2 = K_2 - 1$). The yellow regions of spatial area α_2 are appended to the bottom of sections in the final row ($k_1 = K_1 - 1$). The spatial area overlap of these regions for the final section ($k_1 = K_1 - 1, k_2 = K_2 - 1$) is α_3. These areas are given by

$$\alpha_1 = L_1(K_2 L_2 - N_{m,2}) \tag{5.74}$$

$$\alpha_2 = L_2(K_1 L_1 - N_{m,1}) \tag{5.75}$$

$$\alpha_3 = (K_1 L_1 - N_{m,1})(K_2 L_2 - N_{m,2}) \tag{5.76}$$

As we process each section of the VSM (light blue), we may remove it from the VSM. Assuming row-by-row processing, we may express the spatial area of the rows above section (k_1, k_2) by the expression $k_1 K_2 A_1 - k_1 \alpha_1$. Here, $A_1 = L_1 L_2$. The first term counts the number of sections $(k_1 K_2)$ in the above rows and the second term subtracts the zero padding at the end of these rows, which is not part of the VSM. We may also express the spatial area of the *current* row by the expression $(k_2 + 1)A_1 - I_2\alpha_1 - (k_2 + 1)I_1\alpha_2 + I_1 I_2 \alpha_3$. The first term counts the number of sections $(k_2 + 1)$ in the current row. The second terms subtracts the appropriate area for zero padding if the current section is in the last column. The third term does the same thing for when the current row is the last row. Finally, the last term re-adds in area α_3 if the current section is in both the final row and final column, as in this case the area α_4 would have been subtracted twice. Recall that the functions I_1 and I_2 are given in Eq. 5.38 and 5.39.

We may combine these expressions to determine an expression for the instantaneous optimized OLA VSM,

$$\begin{aligned}\widehat{\text{VSM}}_{OLA}^{r\times r}(k_1,k_2) &= \text{VSM}_{OLA} - B\,(k_1K_2A_1 - k_1\alpha_1 + (k_2+1)A_1 \\ &\quad -I_2\alpha_1 - (k_2+1)I_1\alpha_2 + I_1I_2\alpha_3) \\ &= \text{VSM}_{OLA} - B\,((k_1K_2 + k_2 + 1)A_1 - (k_1+I_2)\alpha_1 - (k_2+1)I_1\alpha_2 + I_1I_2\alpha_3) \end{aligned} \tag{5.77}$$

Note that the initial VSM, prior to processing section $(0,0)$, is equal to VSM_{OLA}. As with the expressions for CSSM and FSSM, $\widehat{\text{VSM}}_{OLA}^{r\times r}(k_1,k_2)$ specifies the VSM after iteration (k_1,k_2) is completed. We may do the same analysis for column-by-column processing. In that case, the instantaneous optimized OLA VSM is given by

$$\begin{aligned}\widehat{\text{VSM}}_{OLA}^{c\times c}(k_1,k_2) &= \text{VSM}_{OLA} - B\,(k_2K_1A_1 - k_2\alpha_2 + (k_1+1)A_1 \\ &\quad -I_1\alpha_2 - (k_1+1)I_2\alpha_1 + I_1I_2\alpha_3) \\ &= \text{VSM}_{OLA} - B\,((k_2K_1 + k_1 + 1)A_1 - (k_2+I_1)\alpha_2 - (k_1+1)I_2\alpha_1 + I_1I_2\alpha_3) \end{aligned} \tag{5.78}$$

We may implement a more memory-efficient form of OLA by clearing VSM memory that can then be used for CSSM and FSSM. In this case, the instantaneous OLA TM is given by

$$\widehat{\text{TM}}_{OLA}^{3D} = \widehat{\text{VSM}}_{OLA}^{r\times r} + \text{CM}_{OLA} + \text{CSSM}^{c\times c} + \text{FSSM}^{c\times c} \tag{5.79}$$

for row-by-row processing and

$$\widehat{\text{TM}}_{OLA}^{3D} = \widehat{\text{VSM}}_{OLA}^{c\times c} + \text{CM}_{OLA} + \text{CSSM}^{c\times c} + \text{FSSM}^{c\times c} \tag{5.80}$$

for column-by-column processing.

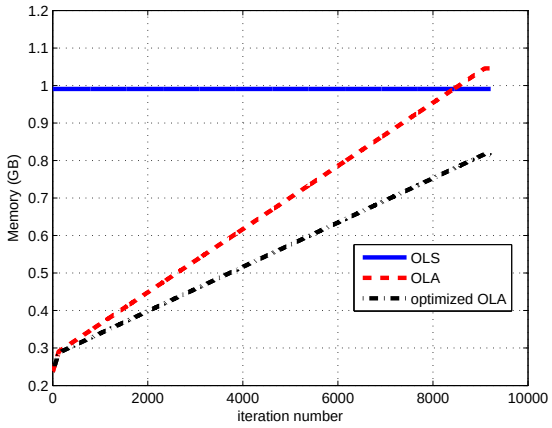
To illustrate this concept at work, we show the instantaneous memory for several examples in Fig. 5.36 for both the first video slice and subsequent video slices. Here, the video size is $N_m = 1080 \times 1920 \times 1000$. We show results for template sizes of $N_h = 50 \times 50 \times 50$ and $N_h = 200 \times 200 \times 200$. Notice in the first case ($N_h = 50 \times 50 \times 50$), the optimized OLA scheme outperforms OLS on the first video slice and has a lower average memory usage than OLS in subsequent slices. However, once the template increases in size ($N_h = 200 \times 200 \times 200$), the optimized OLA scheme does not outperform OLS. Also note that the effects of the CSSM are more significant in this case.

It should be noted that a similar optimization for OLS could be implemented to minimize the average memory required for VSM while processing a video slice. However, because of the overlap between sections, managing this memory becomes considerably more challenging. For example, we may have to wait until the second row is processed before clearing VSM from the first row, because the second row may overlap considerably with the first row. For this reason, we simply point out that the OLS presented thus far can be optimized further to minimize the average memory use. The takeaway message, however, is that OLA will always requires more maximum memory than OLS, which is of significant interest for developing correlation and convolution operations on limited memory platforms.

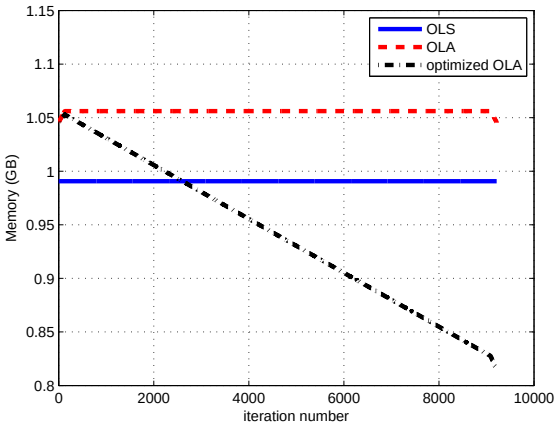
We also emphasize that we have assumed that full frames must be read into VSM. More efficiency can be gained (for both OLA and OLS) by only reading smaller portions of frames into memory at a time (e.g., reading one spatio-temporal video cube into memory at a time).

5.7.1.3 Row-by-Row vs. Column-by-Column Scanning

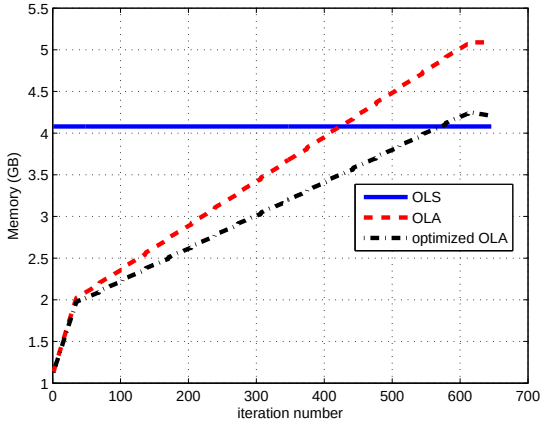
We previously presented expressions for memory for both row-by-row and column-by-column scanning. In each of these cases, the maximum FSSM does not change, and is given by Eq. 5.67 or Eq. 5.70. Because of this, we can choose row-by-row scanning or column-by-column scanning based on the maximum CSSM (because the maximum FSSM does not change). This



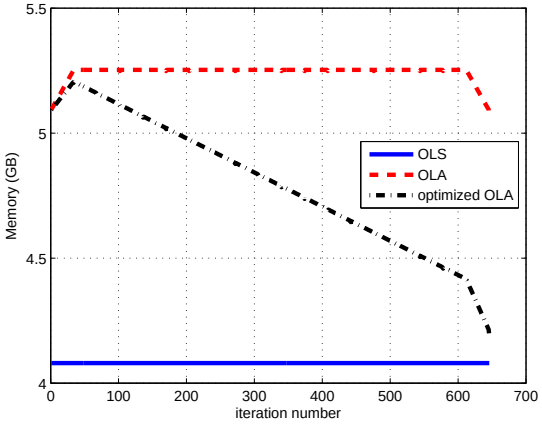
(a) $N_h = 50 \times 50 \times 50$, First Frame



(b) $N_h = 50 \times 50 \times 50$, Subsequent Frame



(c) $N_h = 200 \times 200 \times 200$, First Frame



(d) $N_h = 200 \times 200 \times 200$, Subsequent Frame

Figure 5.36: Demonstration of improved OLA scheme. Here, memory is saved by clearing VSM. In all cases, the video is of size $N_m = 1080 \times 1920 \times 1000$. The template is of size $N_h = 50 \times 50 \times 50$ for A) and B) and of size $N_h = 200 \times 200 \times 200$ for C) and D). In A) and C), the first video slice is depicted, whereas subsequent slices are depicted in C) and D).

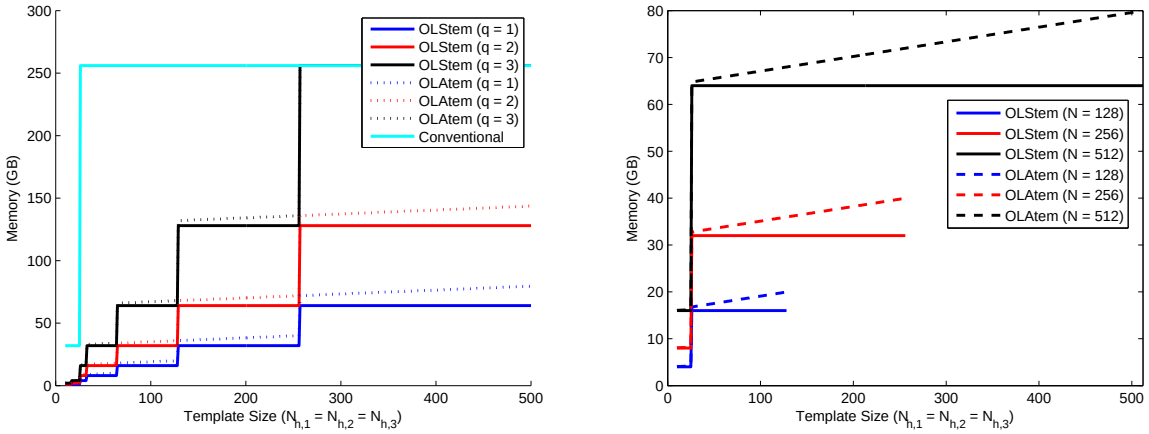
problem, once again, is analogous to the 2D problem. To determine whether to use the row-by-row or column-by-column scanning, we can compare the maximum CSSM for each method. We use row-by-row scanning when

$$\begin{aligned} max(\text{CSSM}^{r\times r}) &< max(\text{CSSM}^{c\times c}) \\ K_2V_2 + V_3 + V_4 &< V_2 + K_1V_3 + V_4 \qquad (5.81) \\ (K_2-1)V_2 &< (K_1-1)V_3 \\ (K_2-1)(N_{h,1}-1)L_2L_3 &< (K_1-1)(N_{h,2}-1)L_1L_3 \\ (K_2-1)(N_{h,1}-1)L_2 &< (K_1-1)(N_{h,2}-1)L_1 \qquad (5.82) \end{aligned}$$

and the column-by-column scanning otherwise. If both sides of the inequality in 5.82 are equal, then either scanning may be used. In the simplest of cases, if $N_{h,1} = N_{h,2}$ and $L_1 = L_2$, then row-by-row scanning should be used when $K_2 < K_1$. By choosing the scanning carefully, memory usage may be reduced.

5.7.2 OLA and OLS in Temporal Dimension Only

In the previous section, we described the application of OLA and OLS in all three dimensions. In this case, we perform OLA and OLS in only the temporal dimension, $d = 3$. This paradigm may be used for when frames must be decompressed and read into memory in their entirety. For OLS, N_3 frames are read into memory, whereas only L_3 frames are read into memory for OLA. In both cases, however, the frames are transformed by a 3D DFT of size $N_{c,1} \times N_{c,2} \times N_3$. Note that we use $N_{c,d}$ with $q_d = 1$ for the first two dimensions, as conventional correlation is performed in these dimensions. Therefore, the total memory elements required for the 3D DFT of each video section is $2N_{c,1}N_{c,2}N_3$. This is the same memory required for the transformed template. Therefore, the most memory intensive step (storing the DFTs of both the template and



(a) Varying q_d

(b) Varying N_d

Figure 5.37: An evaluation of OLA and OLS in only the temporal dimension for videos. We show the effects of varying N_d and the template size. Here, the template is always a square ($N_{h,1} = N_{h,2} = N_{h,3}$) and the video size is $1000 \times 1000 \times 1000$ pixels.

video prior to element-wise multiplication) gives a maximum instantaneous correlation memory of $4N_{c,1}N_{c,2}N_3$.

For OLS, no values need to be stored for the next iteration. However, for OLA, we must store $N_{c,1}N_{c,2}(N_{h,3} - 1)$ elements that represent correlation outputs that must be added to subsequent correlation outputs. Therefore, OLS requires a TM of $4N_{c,1}N_{c,2}N_3$, whereas OLA requires a TM of $4N_{c,1}N_{c,2}N_3 + N_{c,1}N_{c,2}(N_{h,3} - 1)$.

We repeat the evaluations from the previous section and present plots in Fig. 5.37. In the first evaluation, we vary the template size with different values of q_d. Note that OLA and OLS are only performed in the temporal dimension. Comparing this to Fig. 5.32, we see that performing OLA/OLS in all three dimensions offers a substantial reduction in memory required compared to only applying OLA/OLS in the temporal dimension. For example, for a template of size $200 \times 200 \times 200$ and $q_d = 2$, 10.27 GB of memory is needed for OLA in all three dimensions, whereas 70.22 GB is required for OLA in only the temporal direction. In the second evaluation, we vary the template size while keeping the DFT size fixed. This result should be compared to that in Fig. 5.34. Again, we see that it is beneficial to do OLA/OLS in all three dimensions,

rather than just the temporal dimension. For example, for $N = 256$ and a template of size $150 \times 150 \times 150$, OLA and OLS in all three dimensions require 3.10 GB and 2.41 GB respectively. OLA and OLS in the temporal dimension only, on the other hand, require 36.66 GB and 32 GB, respectively.

Clearly, it is best to perform OLA/OLS in all three dimensions, if possible. However, it may be simpler to implement OLA/OLS in only one dimension, at the expense of requiring additional memory.

5.8 Conclusions

In this chapter, we have investigated different methods of correlating a template with a signal of interest. We have demonstrated that OLA and OLS offer significant computational and memory savings over the conventional scheme. In addition, we have shown that there exists a tradeoff between computation and memory for OLA and OLS. We summarize the contributions of this chapter below.

- We presented generalized expressions (for any dimension) for computational complexity for the conventional, OLA, and OLS schemes.
- We illustrated how OLA performs better than OLS computationally, and how this difference is amplified for higher dimensions.
- We experimentally validated our expressions for complexity to show that they are realistic predictors of actual complexity.
- We illustrated how to select OLA and OLS parameters given a template and signal size.
- We illustrated how OLA requires more memory than OLS.
- We described how the scanning direction for OLA can be chosen dynamically to minimize memory usage.
- We illustrated two methods by which OLA and OLS can be applied to videos. One applies

OLA and OLS in all three dimensions, while the other applies OLA and OLS in only the temporal dimension.

- We discussed how memory usage may be reduced by clearing the memory required to store video frames (VSM).
- For both 2D and 3D, we showed how the template size and DFT size affect the amount of required memory.

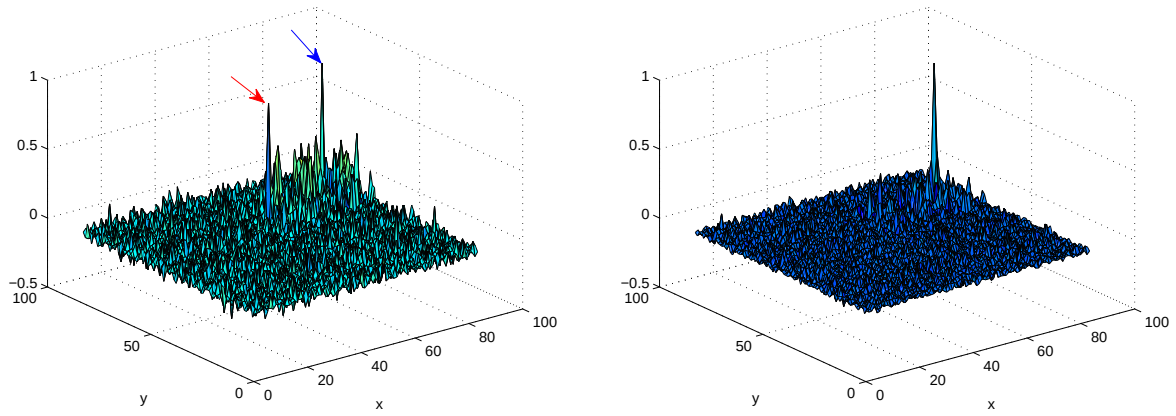
In the earlier chapters, we discussed computational aspects related to the design of CFs. By investigating the DFT size used to design CFs, we determined a serious formulation issue with existing CFs. We fixed this formulation issue by introducing ZACFs in Chapter 3 and the related alternatives in Chapter 4. In this chapter, we discussed the computation and memory issues related to applying the CF to test data. We have shown that we can greatly reduce the computation and memory required by using OLA or OLS. While OLA outperforms OLS computationally, OLS is more efficient from a memory perspective. In the next chapter, we investigate a method to improve OLA to make it more memory efficient and even more computationally efficient. In doing this, we intentionally introduce aliasing into our correlations. Therefore, we will borrow ideas from ZACFs to design CFs that can work in the presence of aliasing.

Chapter 6

Partial-Aliasing Correlation Filters

In the previous chapter, we introduced OLA and OLS, and showed that OLA is more efficient computationally, because it in general needs to process fewer signal sections than OLS. However, OLS holds a significant memory advantage over OLA, because OLS does not need to store the values of output correlations (from previous input sections) to add them to future correlation outputs. It is this computation-memory tradeoff that inspired us to develop a block processing algorithm that attempts to gain both the computational advantages of OLA and the memory advantages of OLS. To do this, we start with OLA and attempt to not only reduce the memory requirements, but also to reduce the computational requirements. This is done by using smaller DFTs. The tradeoff to reducing computation and memory is that we intentionally introduce aliasing into the output correlation plane. To counteract this aliasing, we introduce partial-aliasing CFs (PACFs), which are designed to specifically minimize these aliasing effects. This allows us to achieve a desired correlation output even in the presence of aliasing. For an example, refer to Fig. 6.1. Note that a conventional CF output exhibits aliasing effects, whereas the PACF output resembles the desired output.

In this chapter, we first introduce our new block filtering scheme and the computation and memory advantages it carries. Then, we introduce a method by which to train CFs specifically for this filtering scheme. Finally, we demonstrate results on various types of data.



(a) Conventional CF Output

(b) PACF Output

Figure 6.1: Example outputs from our block filtering scheme that intentionally introduces aliasing. A conventional CF will generate an aliased output, whereas the PACF output will generate an output that closely resembles the desired output. In this example, the correct peak is located at $y = 62$, $x = 73$ and is denoted by a blue arrow. The red arrow indicates a large aliased peak. Note that this aliased peak is very small in the PACF output. In both plots, the correlation outputs have been normalized to unity.

6.1 Partial-Aliasing OLA

Because of the computational advantages of OLA, it is a good starting point for developing an even more efficient block filtering architecture, which we refer to as partial aliasing OLA (PAOLA). PAOLA is designed to reduce the computational and memory requirements of OLA while allowing aliasing. In a sense, PAOLA can be thought of as a generalized version of OLA. Because PAOLA intentionally introduces aliasing into the output correlation, the question is whether CFs can be designed to minimize this aliasing, given the knowledge of the PAOLA architecture. The feasibility of this idea depends on our ability to design CFs that exhibit desired sharp correlation peaks even in the presence of aliasing due to circular correlation. We will turn our attention to filter design in Section 6.2.

We show the PAOLA architecture for 1D in Fig. 6.2a. Consider a signal $\mathbf{x}_q$ of length N_x that is to be correlated with a CF template $\mathbf{h}$ of length N_h. In this case, the input signal is divided into K sections of length L (the last section is zero padded to size L if necessary). Each section

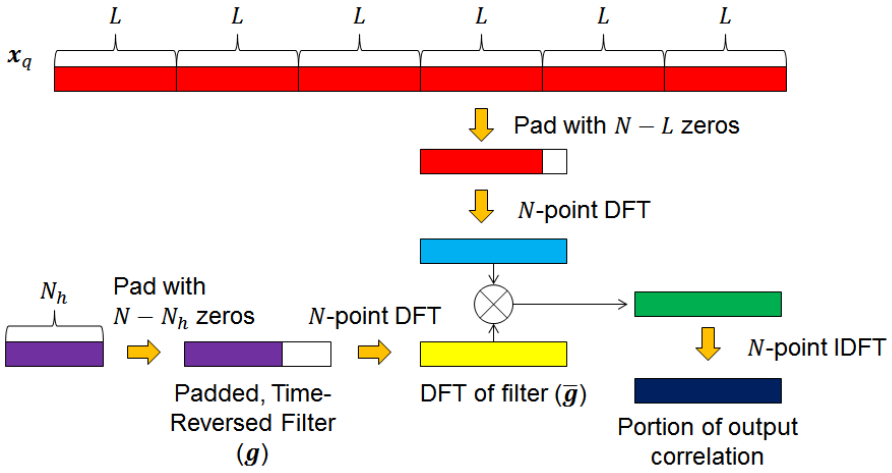
is then padded by $N - L$ zeros and correlated with the CF template using N-point DFTs. If $N < N_h + L - 1$, the correlation result of each section is a circular correlation, and therefore contains aliasing errors. Otherwise, there is no aliasing, and the scheme is equivalent to OLA. The resulting outputs of each section are overlapped appropriately and added together (Fig. 6.2b), generating output $\mathbf{y}_{a,q}$. In our notation, the reason for the subscripts on variables $\mathbf{x}_q$ and $\mathbf{y}_{a,q}$ will become apparent later.

For OLA, we choose the DFT size N to be larger than the template size N_h. Typically, this is done by choosing N to be a power of 2 greater than N_h as described in Eq. 5.5. Then, the cut length L is determined as $L = N + 1 - N_h$ and the number of sections K is determined from the length of the signal N_m and the length of each section. Finally, the number of complex multiplications required to compute the correlation is given by $(2K + 1)f_M(N) + KN$. Recall that an FFT of size N requires $f_M(N) = \frac{N}{2}log_2N$ complex multiplications for a radix-2 FFT. [42].

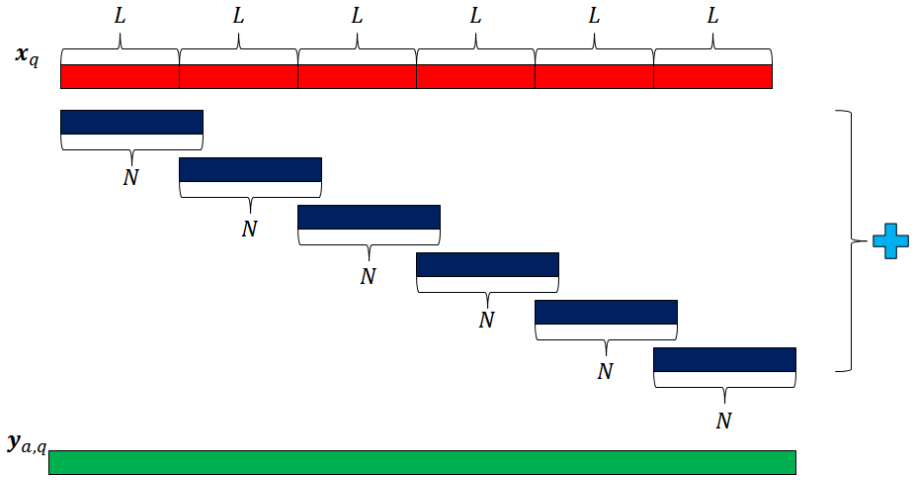
Regardless of what we use for N, using OLA always will mean that $N = L + N_h - 1$. This condition is imposed to ensure that each section correlation generates a linear correlation. For PAOLA, however, we relax this requirement such that $\max(L, N_h) \leq N < L + N_h - 1$. Doing so means that the correlation of each section with the template will be a circular correlation. However, because N is smaller for PAOLA than it is for OLA, we save on computation and memory.

To illustrate this idea, consider a signal of length $N_m = 1000$ and a template of length $N_h = 100$. Using OLA, we can pick an FFT size N that is a power of 2 that is q "steps" above N_h. This choice in N subsequently affects L and K. This is shown for several values of q in Table 6.1. We also show the resulting number of multiplies required for each choice. Finally, we show the number of storage elements required for the OLA storage memory. Recall that for 1D, this is $N_h - 1$ elements, as this is the amount of overlap between sections.

We repeat the same process for PAOLA and show the numbers in Table 6.1. However, we no longer have to choose L to avoid circular correlation. Therefore, we can make L larger than



(a) Processing the signal one section at a time



(b) Combining outputs of each section

Figure 6.2: An illustration of the PAOLA architecture. Like OLA, each non-overlapping section is processed one at a time

Table 6.1: A Comparison of OLA and PAOLA

	q	N	L	K	Number of Multiplications	Storage Memory Elements
OLA	1	128	29	35	36,288	99
	2	256	157	7	**17,152**	99
	3	512	413	3	17,664	99
	4	1024	925	2	27,648	99
PAOLA	1	128	50	20	20,928	78
			100	10	10,688	28
	2	256	200	5	12,544	56
			250	4	**10,240**	**6**
	3	512	500	2	12,544	12

the corresponding value of L used for OLA. Doing this decreases K, but increases the amount of aliasing due to circular correlation. Note that L can be chosen as a factor of the signal length N_m to avoid a partial section shorter than L at the end of the signal. Because L is larger, the corresponding storage memory (now given by $N - L$) is smaller.[1] Note that we can achieve much greater efficiency (in terms of computation and memory) by leveraging PAOLA.

Another way to think about PAOLA vs. OLA is to consider a system that has a fixed section length L, which means that the number of sections K for a given signal cannot change. This makes the DFT size the primary factor determining computational complexity. In this case, PAOLA can use a smaller DFT size than OLA to process each section, which would subsequently reduce computation and memory requirements. For more detail on the computational and memory benefits of PAOLA, see Section 6.1.2.

6.1.1 PAOLA Example

In correlation pattern recognition, circular correlation effects are acceptable provided that the correlation peak is preserved and is in the correct location. To demonstrate circular correlation effects generated by the PAOLA architecture, we present an example. In this example, the signal of interest is a 201-sample chirp signal. In this case, we train a CF using 10 aligned training

[1]Note that in the OLA case, $N_h - 1 = N - L$. For PAOLA, this equality does not hold because it allows circular correlation.

signals. The CF is of length $N_h = 201$ and is trained using a DFT of size[2] $N = 201$. Note that we do not use a ZACF for this example.

We set the cut length as $L = 201$. In Fig. 6.3, we show the test signal, which contains a chirp that starts at time index 25 and ends at time index 225. An ideal output would be a sharp peak located at time index 225. We will process this test signal using both OLA ($N = L + N_h - 1 = 401$) and PAOLA ($N = 201$).

In Fig. 6.4a, we show the output of PAOLA processing the first section ($k = 1$, indices 1 to 201). Note that there is an erroneous peak at index 24 (recall that the filter should output a peak at the end of the chirp instead of the beginning of the chirp). We show the output of OLA processing the same section in Fig. 6.4b. Note that there is a correct peak at index 225. Note that if we were take this output and add indices 202 to 401 to indices 1 to 200, we would get the same thing as the PAOLA output in Fig. 6.4a. Therefore, cutting the chirp into two pieces (the first half located in section $k = 1$ and the second half located in section $k = 2$) has created a false, aliased peak at index 24. In Fig. 6.4c, we show the output of PAOLA processing section $k = 2$. Due to the tail portion of the chirp, we get a small peak located at the correct location, 225. As before, this output is an aliased version of the OLA output shown in Fig. 6.4d. However, in this case, the aliased values are small and therefore do not affect the peak much. Finally, we show the full output for PAOLA in Fig. 6.4e and the full output for OLA in Fig. 6.4f. Note that there are two peaks for the PAOLA, with the larger of the two peaks being incorrect. For OLA, there is only one peak at the correct location. This effect is just one of the possible outcomes that we observe. If the first section that includes the chirp only includes a small part of the chirp, the erroneous peak will be very low, and the true peak will be higher. In addition, this effect lessens as the DFT size approaches $N_d = 2N_{h,d} - 1$.

To combat these effects, we develop an enhanced technique for CF design. We refer to this class of filters as PACFs. After developing this scheme, we will test it against baseline filters to

[2]We will not always use efficient (i.e., a power of 2) DFT sizes in this example, for the purpose of showing the continuum of performance as the DFT size is varied. In an actual application, the DFT size should be chosen such that the algorithm of choice (e.g. FFT) is efficient.

illustrate how performance changes with DFT size.

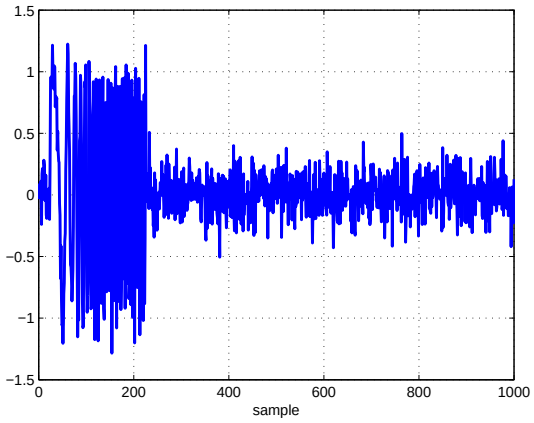


Figure 6.3: Example signal with a chirp starting at time index 25 and ending at time index 225.

6.1.2 Computation and Memory for PAOLA in Higher Dimensions

Previously, we gave a brief explanation of the benefits of PAOLA in 1D. To be complete, we present the details of the computational and memory requirements for PAOLA in higher dimensions.

6.1.2.1 Computation

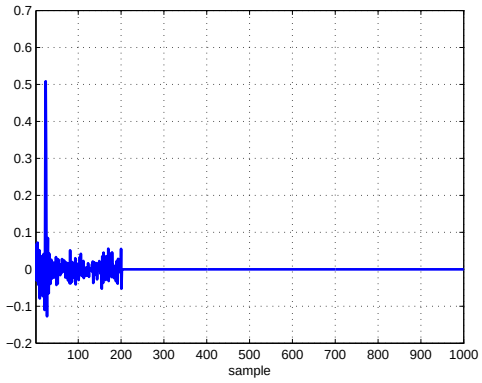
Like OLA and OLS, the number of computations are given by Eq. 5.20, reprinted here for convenience,

$$C_{OLX,d}(N_d, K_d) = \left(1 + 2K^{\#}\right) \left[\sum_{d=1}^{D} \frac{f_M(N_d)}{N_d} N^{\#}\right] + K^{\#} N^{\#} \tag{6.1}$$

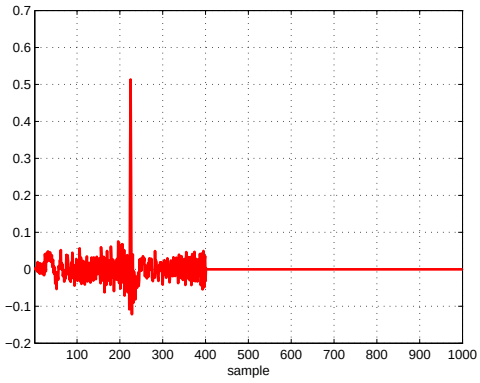
The number of sections in dimension d is given by

$$K_{PAOLA,d} = \left\lceil \frac{N_{m,d}}{L_d} \right\rceil \tag{6.2}$$

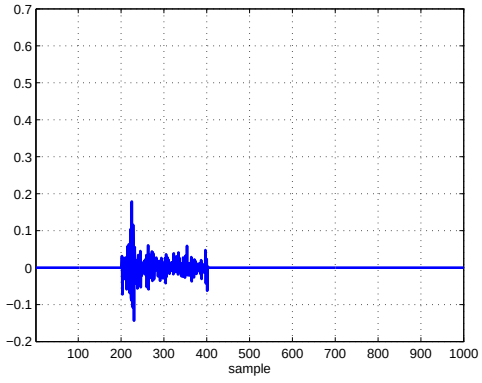
as in the OLA case. In Eq. 6.1, $K^{\#} = \prod_{d=1}^{D} K_{PAOLA,d}$.



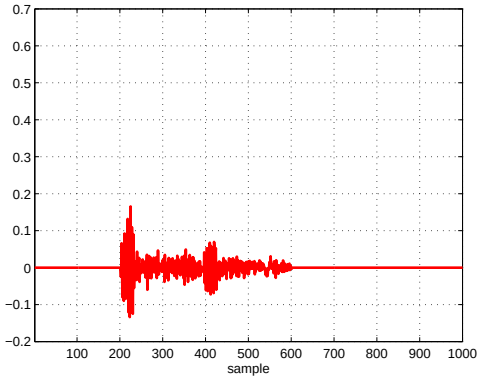
(a) PAOLA, $k = 1$



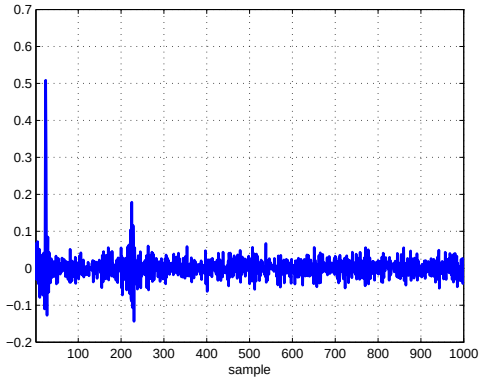
(b) OLA, $k = 1$



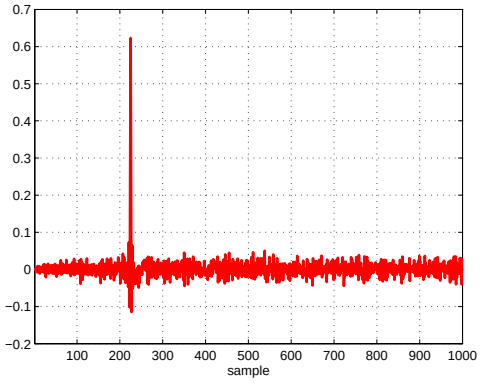
(c) PAOLA, $k = 2$



(d) OLA, $k = 2$



(e) PAOLA, Complete Output



(f) OLA, Complete Output

Figure 6.4: Example of circular correlation effects in a PAOLA architecture. See text for details.

In the previous chapter, we showed that OLA outperformed OLS from a computational perspective because it required processing fewer sections. For PAOLA, L_d and N_d are chosen such that $N_d < L_d + N_{h,d} - 1$, whereas OLA requires that $N_d = L_d + N_{h,d} - 1$. PAOLA outperforms OLA via two mechanisms. First, if L_d is held constant, N_d can be chosen to be smaller than the DFT size used for OLA. Second, if N_d is held constant, the section length L_d for PAOLA can be longer than that of OLA (this results in processing fewer sections). However, both of these mechanisms cause increased aliasing due to circular correlation.

6.1.2.2 Memory

PAOLA also requires less memory than that of OLA. In 2D, recall that OLA requires both correlation memory (CM) and storage memory (SM). If we fix N_d, OLA and PAOLA will require the same CM. However, PAOLA allows us to use a larger L_d. This decreases the SM necessary. To see this, refer to Fig. 6.5, where we compare the memory regions for OLA and PAOLA. Note that the number of elements in regions 2, 3, and 4 for OLA are given in Eq. 5.34 through 5.36. For PAOLA, these areas are given by

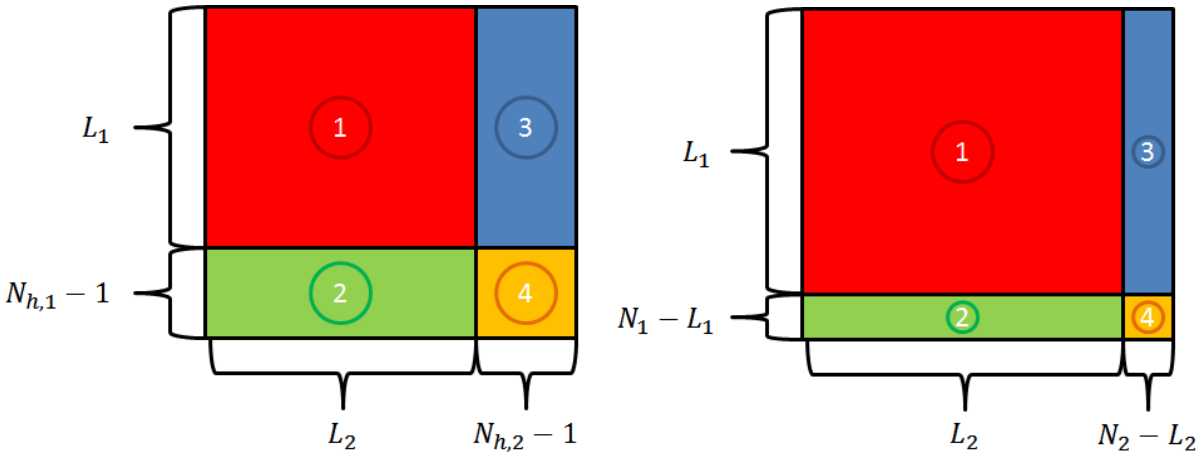
$$A_2 = (N_1 - L_1)L_2 \tag{6.3}$$

$$A_3 = (N_2 - L_2)L_1 \tag{6.4}$$

$$A_4 = (N_1 - L_1)(N_2 - L_2) \tag{6.5}$$

With these new expressions for A_2, A_3, and A_4, the memory analysis for OLA performed in Chapter 5 applies to PAOLA as well. Note that as L_1 and L_2 increase, the SM component of the total memory decreases. At the limit, $L_1 = N_1$ and $L_2 = N_2$, and the SM is zero. Therefore, PAOLA can offer a significant memory advantage over OLA. However, saving this memory incurs the cost of increased aliasing.

In 6.6, we illustrate the difference between OLA and PAOLA in 3D. The number of elements



(a) OLA (b) PAOLA

Figure 6.5: Regions of memory for one output section correlation in 2D, OLA vs. PAOLA. Note that L_1 and L_2 are different for OLA and PAOLA.

in the volumes 2-8 for OLA are given in Eq. 5.52 through 5.58. For PAOLA, these volumes are given by

$$V_2 = (N_1 - L_1)L_2L_3 \tag{6.6}$$

$$V_3 = (N_2 - L_2)L_1L_3 \tag{6.7}$$

$$V_4 = (N_1 - L_1)(N_2 - L_2)L_3 \tag{6.8}$$

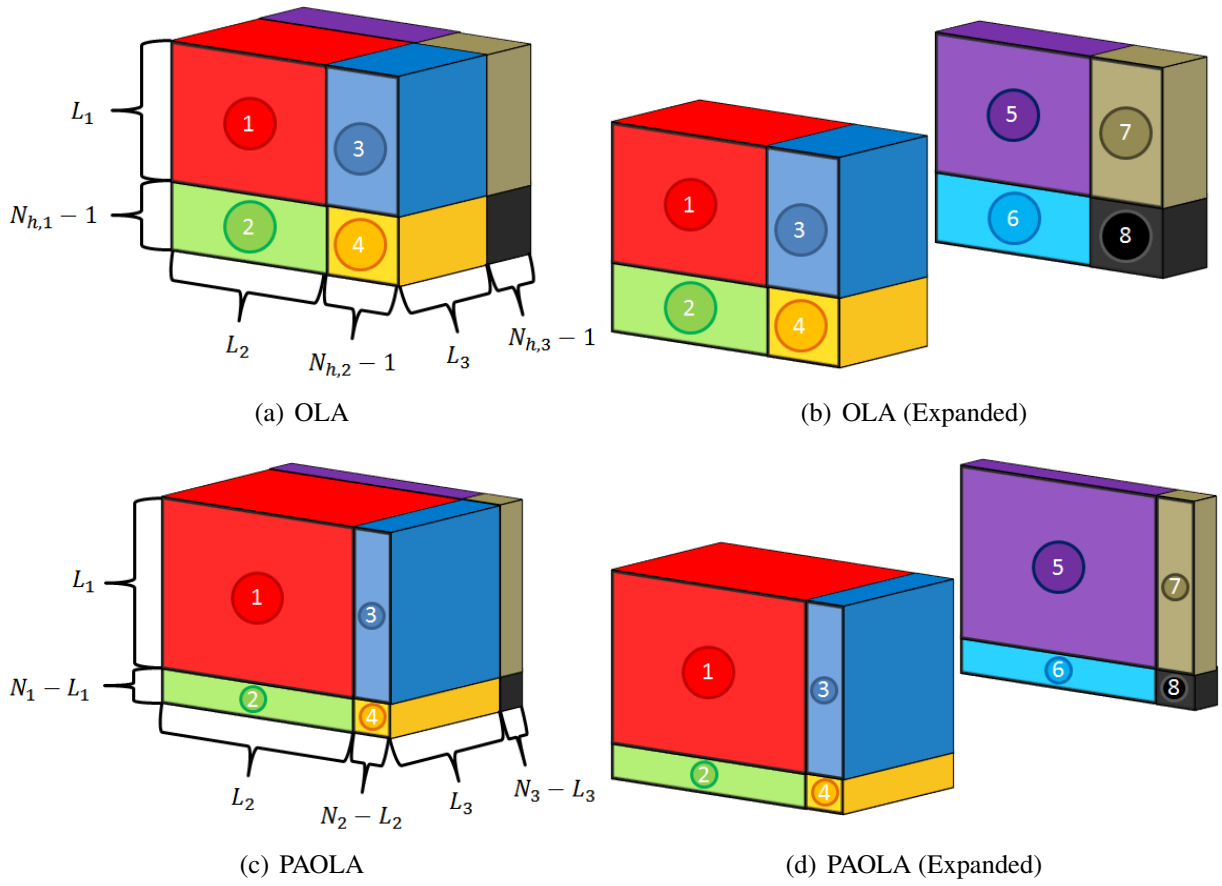
$$V_5 = L_1L_2(N_3 - L_3) \tag{6.9}$$

$$V_6 = (N_1 - L_1)(N_3 - L_3)L_2 \tag{6.10}$$

$$V_7 = (N_2 - L_2)(N_3 - L_3)L_1 \tag{6.11}$$

$$V_8 = (N_1 - L_1)(N_2 - L_2)(N_3 - L_3) \tag{6.12}$$

With these new expressions for V_2 through V_8, the memory analysis for OLA performed in Chapter 5 applies to PAOLA as well. Recall that the memory requirement is a function of CM, video slice memory (VSM), current slice storage memory (CSSM), and future slice storage memory



(a) OLA (b) OLA (Expanded)

(c) PAOLA (d) PAOLA (Expanded)

Figure 6.6: Regions of memory for one output section correlation in 3D, OLA vs. PAOLA. Note that L_1, L_2, and L_3 are different for OLA and PAOLA.

(FSSM). If N_d is fixed, the CM is the same for OLA and PAOLA. However, CSSM (which is similar to SM in 2D) decreases. In addition, FSSM decreases because $N_3 - L_3 < N_{h,3} - 1$ for PAOLA. Finally, note that VSM for PAOLA is actually larger than it is for OLA. This is because the cut length L_3 is longer, so more frames must be read into memory at once. This is, of course, a free parameter, and one can always use a smaller value of L_3 to reduce VSM and simply reap the benefits of PAOLA in the other dimensions instead. If we go to the limit ($L_1 = N_1$, $L_2 = N_2$, and $L_3 = N_3$), VSM is maximum (and equal to VSM for OLS), but CSSM and FSSM approach zero.

For both the 2D and 3D cases, the amount of memory saved is dependent on how much aliasing is considered acceptable. Therefore, we do not explicitly quote figures regarding the amount of memory saved for PAOLA vs. OLA. It is important to note that PAOLA will, at best, perform the same as OLS in terms of memory (when $L_d = N_d$). At worse, PAOLA requires the same amount of memory as OLA (when $L_d = N_d + 1 - N_{h,d}$).

6.2 PACF Formulation

Our formulation of PACFs was inspired by [59], which was aimed at the design of frequency-selective filters (e.g., an ideal band-pass filter) for OLA architectures. The filter inputs in [59] are modeled as sample realizations of white noise; in our application, the filter inputs are a set of specific training signals and cannot be modeled as white noise. In addition, filter phase for CFs is more important [63] than the filter magnitude, i.e., CFs are not frequency-selective filters. In this section, we will first discuss the typical method used to train a CF. Then, we will explain the PACF formulation. We use 1D notation to introduce the PACF formulation. We later explain how it is extended to higher dimensions.

In this section, we use a slightly different notational convention. First, we define $\bar{\mathbf{g}}$ as the N-point DFT of the time-reversed template $\mathbf{h}$,

$$\bar{\mathbf{g}} = \mathbf{F}_N[flip(\mathbf{h})^T \ \mathbf{0}_{N-N_h}^T]^T \tag{6.13}$$

where the operation $flip()$ flips the CF template[3]. Recall that $\mathbf{F}_N$ and $\mathbf{F}_N^{-1}$ denote the $N \times N$ DFT and IDFT matrices, respectively. In addition, we will train a baseline CF using Q cropped signals $\mathbf{s}_q$, $q = 1, \ldots, Q$, of length N_s. For example, if we were to train a CF for the chirp found in Fig. 6.3, each $\mathbf{s}_q$ would be a $N_s = 201$ sample chirp. For PACF, we will train using Q signals $\mathbf{x}_q$, $q = 1, \ldots, Q$, of size $N_x > N_s$. Typically, $\mathbf{x}_q$ would contain the embedded signal of interest, $\mathbf{s}_q$. We use these longer training signals to account for interactions between each section when block filtering the signal using PAOLA.

6.2.1 Baseline (MOSSE) Filter

Although we discussed the MOSSE filter and ZAMOSSE filter in Section 3.6.3 and 3.6.4, respectively, we present it here for clarity using the notation that we will use for PACFs. The MOSSE filter is designed to minimize an error function

$$e = \sum_{q=1}^{Q} \left\| \bar{\mathbf{S}}_q \bar{\mathbf{g}} - \bar{\mathbf{y}}_{d,q} \right\|^2 = \sum_{q=1}^{Q} (\bar{\mathbf{S}}_q \bar{\mathbf{g}} - \bar{\mathbf{y}}_{d,q})^+ (\bar{\mathbf{S}}_q \bar{\mathbf{g}} - \bar{\mathbf{y}}_{d,q}) \tag{6.14}$$

where $\bar{\mathbf{y}}_{d,q}$ is the N point DFT of the desired correlation output. Here, $\bar{\mathbf{S}}_q = diag(\mathbf{F}_N \mathbf{s}_q)$ is an $N \times N$ matrix with the N-point DFT of $\mathbf{s}_q$ along its diagonal The product $\bar{\mathbf{S}}_q \bar{\mathbf{g}}$ represents the correlation[4] of $\mathbf{s}_q$ with the CF template in the frequency domain. Minimizing Eq. 6.14 with respect to $\bar{\mathbf{g}}$ yields

[3]This notation is in place to facilitate implementation. We find that flipping the template prior to DFT (as opposed to taking the conjugate in the frequency domain) leads to a more straightforward implementation of PAOLA in tools such as MATLAB.

[4]This is technically a circular correlation, as the first part of this thesis demonstrates. Zero-aliasing constraints will ensure that the *training* process is free from circular correlation effects. In normal filtering methods that avoid aliasing at test time (e.g. OLA) a filter trained with these constraints will perform better. However, in the PAOLA case (which *does* allow aliasing), we found that the traditional MOSSE filter performs slightly better than its zero-aliasing counterpart, so we use it as the baseline here.

$$\bar{\mathbf{g}} = \mathbf{P}_{ss}^{-1}\mathbf{p}_{sy} \tag{6.15}$$

where

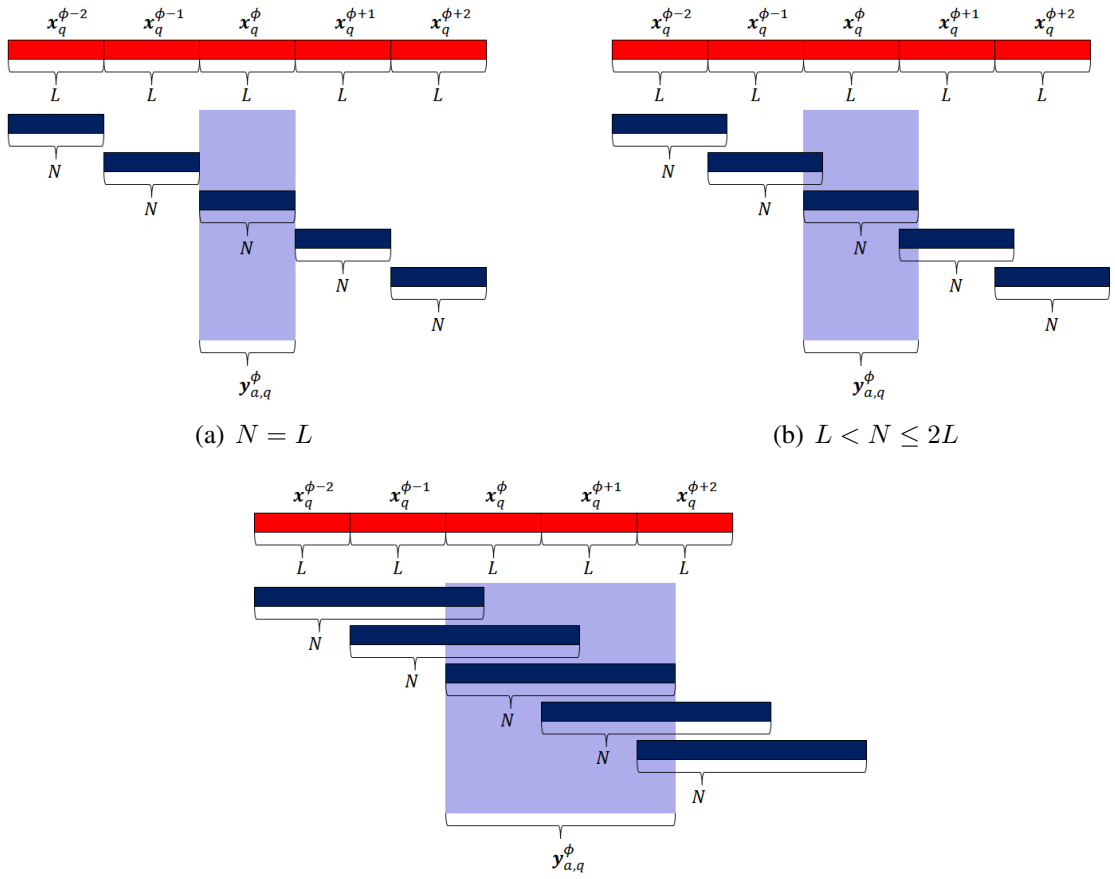
$$\mathbf{P}_{ss} = \sum_{q=1}^{Q} \bar{\mathbf{S}}_q^{+}\bar{\mathbf{S}}_q \tag{6.16}$$

$$\mathbf{p}_{sy} = \sum_{q=1}^{Q} \bar{\mathbf{S}}_q^{+}\bar{\mathbf{y}}_{d,q} \tag{6.17}$$

This expression is written in matrix-vector notation, but is equivalent to the expression in [21]. Note that this method does not make any assumptions about the structure of the correlation architecture used to apply the filter (i.e. it does not account for the aliasing effects that are present in PAOLA).

6.2.2 PACF

We now formulate the PACF, which takes into account the structure of PAOLA. First, we represent $\mathbf{x}_q^{\phi}$ as the ϕth section (L samples long) of training signal $\mathbf{x}_q$. In a similar fashion, $\mathbf{y}_{a,q}^{\phi}$ and $\mathbf{y}_{d,q}^{\phi}$ represent the ϕth section of the actual output $\mathbf{y}_{a,q}$ and the desired output $\mathbf{y}_{d,q}$, respectively, and that both are N samples long. Note that $\mathbf{y}_{a,q}^{\phi}$ is not simply the correlation output of one input section; rather, it is the final summed output of all overlapping sections (see Fig. 6.7). In the frequency domain, we can formulate the corresponding error metric as follows.



(a) $N = L$ (b) $L < N \leq 2L$

(c) $N > 2L$

Figure 6.7: Different situations based on the size of the DFT used for block correlation filtering.

$$
\begin{aligned}
e &= \sum_{q=1}^{Q} \sum_{\phi=1}^{K} \left\| \bar{\mathbf{y}}_{a,q}^{\phi} - \bar{\mathbf{y}}_{d,q}^{\phi} \right\|^2 \\
&= \sum_{q=1}^{Q} \sum_{\phi=1}^{K} (\bar{\mathbf{y}}_{a,q}^{\phi} - \bar{\mathbf{y}}_{d,q}^{\phi})^{+} (\bar{\mathbf{y}}_{a,q}^{\phi} - \bar{\mathbf{y}}_{d,q}^{\phi})
\end{aligned} \tag{6.18}
$$

Note that $\bar{\mathbf{y}}_{a,q}^{\phi} = \mathbf{F}_N \mathbf{y}_{a,q}^{\phi}$ and $\bar{\mathbf{y}}_{d,q}^{\phi} = \mathbf{F}_N \mathbf{y}_{d,q}^{\phi}$. In a similar fashion as before, $\bar{\mathbf{X}}_q^{\phi} = diag(\mathbf{F}_N \mathbf{x}_q^{\phi})$ is an $N \times N$ matrix with the N-point DFT of $\mathbf{x}_q^{\phi}$ along its diagonal. The frequency domain representation of the (circular) correlation of section $\mathbf{x}_q^{\phi}$ with the CF is then given by $\bar{\mathbf{X}}_q^{\phi} \bar{\mathbf{g}}$.

To solve for the optimal CF, we must derive an expression for the actual correlation output

of section ϕ, $\bar{\mathbf{y}}_{a,q}^{\phi}$. First of all, the output correlation section $\bar{\mathbf{y}}_{a,q}^{\phi}$ will, in general, be a function of more than one input sections as illustrated in Fig. 6.7. Note that if $N = L$ (Fig. 6.7a), there is no overlap, and output section $\bar{\mathbf{y}}_{a,q}^{\phi}$ only depends on input section $\bar{\mathbf{x}}_{q}^{\phi}$. In the second case ($L < N \leq 2L$) (Fig. 6.7b), the output section $\bar{\mathbf{y}}_{a,q}^{\phi}$ is a function of the current section $\bar{\mathbf{x}}_{q}^{\phi}$ as well as both the previous section $\bar{\mathbf{x}}_{q}^{\phi-1}$ and the next section $\bar{\mathbf{x}}_{q}^{\phi+1}$. For the last case ($N > 2L$) (Fig. 6.7c), the output section $\bar{\mathbf{y}}_{a,q}^{\phi}$ is a function of the current section $\bar{\mathbf{x}}_{q}^{\phi}$ as well as multiple previous and multiple future sections.

Suppose we are processing section ϕ. Then, we define the amount of overlap for a future (i.e. $\phi + \phi^{'}$) or past (i.e. $\phi - \phi^{'}$) section as

$$\sigma(\phi^{'}) = \max(0,\ N - \phi^{'} L) \tag{6.19}$$

where $\phi^{'}$ is a positive integer. This is the same function for both future and past sections because of symmetry. If the term $N - \phi^{'} L$ is negative, there is no overlap, and the function $\sigma(\phi^{'})$ returns 0. Next, we find the highest value of $\phi^{'}$ in which section $\phi \pm \phi^{'}$ will overlap section ϕ,

$$\phi^{'}_{max} = \max \phi^{'} \qquad | \ \sigma(\phi^{'}) > 0 \tag{6.20}$$

Note that the *last* $\sigma(\phi^{'})$ samples of *previous* sections $\phi - \phi^{'}$ overlap with the *first* $\sigma(\phi^{'})$ samples of section ϕ. Similarly, the *first* $\sigma(\phi^{'})$ samples of *future* sections $\phi + \phi^{'}$ overlap with the *last* $\sigma(\phi^{'})$ samples of section ϕ. To express this, we define the following $N \times N$ matrix $\check{\mathbf{R}}_{B}^{N}$, which selects the first B elements of an $N \times 1$ vector and places them at the end of the output vector,

$$\check{\mathbf{R}}_{B}^{N} = \begin{bmatrix} \mathbf{0}_{(N-B)\times B} & \mathbf{0}_{(N-B)\times(N-B)} \\ \mathbf{I}_{B} & \mathbf{0}_{B\times(N-B)} \end{bmatrix} \tag{6.21}$$

For example, if $B = 2$ and $N = 4$, $\check{\mathbf{R}}_{B}^{N}[\ 1 \quad 2 \quad 3 \quad 4\ ]^{T} = [\ 0 \quad 0 \quad 1 \quad 2\ ]^{T}$. Similarly, we define the following $N \times N$ matrix $\hat{\mathbf{R}}_{B}^{N}$, which selects the last B elements of an $N \times 1$ vector and places

them at the front of the output vector,

$$\hat{\mathbf{R}}_B^N = \begin{bmatrix} \mathbf{0}_{B\times(N-B)} & \mathbf{I}_B \\ \mathbf{0}_{(N-B)\times(N-B)} & \mathbf{0}_{(N-B)\times B} \end{bmatrix} \tag{6.22}$$

For example, if $B = 2$ and $N = 4$, $\hat{\mathbf{R}}_B^N[\ 1 \quad 2 \quad 3 \quad 4\ ]^T = [\ 3 \quad 4 \quad 0 \quad 0\ ]^T$. We may now write the frequency domain expression for output section ϕ as follows.

$$\begin{aligned} \bar{\mathbf{y}}_{a,q}^{\phi} &= \bar{\mathbf{X}}_q^{\phi}\bar{\mathbf{g}} + \sum_{\phi'=1}^{\phi'_{max}} \mathbf{F}_N \hat{\mathbf{R}}_{\sigma(\phi')}^N \mathbf{F}_N^{-1} \bar{\mathbf{X}}_q^{\phi-\phi'}\bar{\mathbf{g}} + \sum_{\phi'=1}^{\phi'_{max}} \mathbf{F}_N \check{\mathbf{R}}_{\sigma(\phi')}^N \mathbf{F}_N^{-1} \bar{\mathbf{X}}_q^{\phi+\phi'}\bar{\mathbf{g}} \\ &= \left(\bar{\mathbf{X}}_q^{\phi} + \sum_{\phi'=1}^{\phi'_{max}} \mathbf{F}_N \hat{\mathbf{R}}_{\sigma(\phi')}^N \mathbf{F}_N^{-1} \bar{\mathbf{X}}_q^{\phi-\phi'} + \sum_{\phi'=1}^{\phi'_{max}} \mathbf{F}_N \check{\mathbf{R}}_{\sigma(\phi')}^N \mathbf{F}_N^{-1} \bar{\mathbf{X}}_q^{\phi+\phi'} \right) \bar{\mathbf{g}} \\ &= \mathbf{J}_q^{\phi}\bar{\mathbf{g}} \end{aligned} \tag{6.23}$$

where

$$\mathbf{J}_q^{\phi} \;=\; \bar{\mathbf{X}}_q^{\phi} + \sum_{\phi'=1}^{\phi'_{max}} \left(\mathbf{F}_N \hat{\mathbf{R}}_{\sigma(\phi')}^N \mathbf{F}_N^{-1} \bar{\mathbf{X}}_q^{\phi-\phi'} + \mathbf{F}_N \check{\mathbf{R}}_{\sigma(\phi')}^N \mathbf{F}_N^{-1} \bar{\mathbf{X}}_q^{\phi+\phi'} \right) \tag{6.24}$$

Note that the above equation does not explicitly account for edge effects at the beginning and the end of the complete input signal. In such cases we simply assume that $\mathbf{X}_q^{\phi\pm\phi'}$ is zero for sections that do not exist (for example, the first section will not have any preceding sections overlapping it, regardless of the choice of N, N_h, or L).

Next, we present another design option: the ability to constrain the length of the template. Here, we use the zero-aliasing constraints developed in [41] to restrict the template length to $N_h \leq N_s$. Doing this will further reduce or eliminate aliasing effects. These constraints force the last $N - N_h$ elements of $\mathbf{g}$ (time domain) to be equal to zero. These constraints are given by

$$\mathbf{A}^{+}\bar{\mathbf{g}} = \mathbf{0} \tag{6.25}$$

where the matrix $\mathbf{A}^{+}$ (Eq. 3.4) is made up of the last $N - N_h$ rows of $\mathbf{F}_N^{-1}$. We now substitute Eq. 6.23 into the error metric e (Eq. 6.18) and minimize it with respect to $\bar{\mathbf{g}}$, subject to the constraints in Eq. 6.25,

$$\begin{aligned}
&\min_{\bar{\mathbf{g}}} \sum_{q=1}^{Q}\sum_{\phi=1}^{K} (\mathbf{J}_q^{\phi}\bar{\mathbf{g}} - \bar{\mathbf{y}}_{d,q}^{\phi})^{+}(\mathbf{J}_q^{\phi}\bar{\mathbf{g}} - \bar{\mathbf{y}}_{d,q}^{\phi}) - \boldsymbol{\omega}^{+}\mathbf{A}^{+}\bar{\mathbf{g}} \\
&\min_{\bar{\mathbf{g}}} \sum_{q=1}^{Q}\sum_{\phi=1}^{K} \left(\bar{\mathbf{g}}^{+}\mathbf{J}_q^{\phi+}\mathbf{J}_q^{\phi}\bar{\mathbf{g}} - \bar{\mathbf{g}}^{+}\mathbf{J}_q^{\phi+}\bar{\mathbf{y}}_{d,q}^{\phi} - \bar{\mathbf{y}}_{d,q}^{\phi+}\mathbf{J}_q^{\phi}\bar{\mathbf{g}} + \bar{\mathbf{y}}_{d,q}^{\phi+}\bar{\mathbf{y}}_{d,q}^{\phi} \right) - \boldsymbol{\omega}^{+}\mathbf{A}^{+}\bar{\mathbf{g}}
\end{aligned} \tag{6.26}$$

where $\boldsymbol{\omega}$ is a Lagrange multiplier vector. If we differentiate this expression with respect to $\bar{\mathbf{g}}$, and set it to $\mathbf{0}$, we obtain

$$\begin{aligned}
\mathbf{0} &= \sum_{q=1}^{Q}\sum_{\phi=1}^{K} \left(\mathbf{J}_q^{\phi+}\mathbf{J}_q^{\phi}\bar{\mathbf{g}} - \mathbf{J}_q^{\phi+}\bar{\mathbf{y}}_{d,q}^{\phi} \right) - \mathbf{A}\boldsymbol{\omega} \\
\left(\sum_{q=1}^{Q}\sum_{\phi=1}^{K} \mathbf{J}_q^{\phi+}\mathbf{J}_q^{\phi} \right) \bar{\mathbf{g}} &= \left(\sum_{q=1}^{Q}\sum_{\phi=1}^{K} \mathbf{J}_q^{\phi+}\bar{\mathbf{y}}_{d,q}^{\phi} \right) + \mathbf{A}\boldsymbol{\omega} \\
\mathbf{P}_{jj}\bar{\mathbf{g}} &= \mathbf{p}_{jy} + \mathbf{A}\boldsymbol{\omega} \\
\bar{\mathbf{g}} &= \mathbf{P}_{jj}^{-1}(\mathbf{p}_{jy} + \mathbf{A}\boldsymbol{\omega})
\end{aligned} \tag{6.27}$$

where

$$\mathbf{P}_{jj} = \sum_{q=1}^{Q} \sum_{\phi=1}^{K} \mathbf{J}_q^{\phi+} \mathbf{J}_q^{\phi} \tag{6.28}$$

$$\mathbf{p}_{jy} = \sum_{q=1}^{Q} \sum_{\phi=1}^{K} \mathbf{J}_q^{\phi+} \bar{\mathbf{y}}_{d,q}^{\phi} \tag{6.29}$$

We assume that $\mathbf{P}_{jj}$ is invertible. We now substitute Eq. 6.27 into Eq. 6.25,

$$\begin{aligned} \mathbf{A}^{+}\mathbf{P}_{jj}^{-1}(\mathbf{p}_{jy} + \mathbf{A}\boldsymbol{\omega}) &= \mathbf{0} \\ \mathbf{A}^{+}\mathbf{P}_{jj}^{-1}\mathbf{A}\boldsymbol{\omega} &= -\mathbf{A}^{+}\mathbf{P}_{jj}^{-1}\mathbf{p}_{jy} \\ \boldsymbol{\omega} &= -\left(\mathbf{A}^{+}\mathbf{P}_{jj}^{-1}\mathbf{A}\right)^{-1}\left(\mathbf{A}^{+}\mathbf{P}_{jj}^{-1}\mathbf{p}_{jy}\right) \end{aligned} \tag{6.30}$$

Substituting this into Eq. 6.27, we obtain our final solution as

$$\bar{\mathbf{g}} = \mathbf{P}_{jj}^{-1}\mathbf{p}_{jy} - \mathbf{P}_{jj}^{-1}\mathbf{A}\left(\mathbf{A}^{+}\mathbf{P}_{jj}^{-1}\mathbf{A}\right)^{-1}\left(\mathbf{A}^{+}\mathbf{P}_{jj}^{-1}\mathbf{p}_{jy}\right) \tag{6.31}$$

We can then find the time domain CF template as

$$\mathbf{h} = flip(\tilde{\mathbf{S}}\mathbf{F}_N^{-1}\bar{\mathbf{g}}) \tag{6.32}$$

where the $N_h \times N$ matrix $\tilde{\mathbf{S}}$ (defined below) selects the first N_h elements the $N \times 1$ vector $\mathbf{g}$, (such that $\tilde{\mathbf{S}}\mathbf{g}$ is an $N_h \times 1$ vector),

$$\tilde{\mathbf{S}} = \begin{bmatrix} \mathbf{I}_{N_h} & \mathbf{0}_{N_h \times N - N_h} \end{bmatrix} \tag{6.33}$$

Note that $\mathbf{I}_{N_h}$ is an $N_h \times N_h$ identity matrix.

Special cases. Note that there are several special cases of the result in Eq. 6.31. If $N_h = N$, then

the constraints are not needed, and our solution is simply

$$\bar{\mathbf{g}} = \mathbf{P}_{jj}^{-1}\mathbf{p}_{jy} \tag{6.34}$$

If we set $N = L = N_h$, there will be no overlap between adjacent sections. In this case, $\mathbf{J}_q^{\phi} = \bar{\mathbf{X}}_q^{\phi}$, and the expression is greatly reduced to

$$\bar{\mathbf{g}} = \mathbf{P}_{xx}^{-1}\mathbf{p}_{xy} \tag{6.35}$$

where

$$\mathbf{P}_{xx} = \sum_{q=1}^{Q}\sum_{\phi=1}^{K}\bar{\mathbf{X}}_q^{\phi+}\bar{\mathbf{X}}_q^{\phi} \tag{6.36}$$

is a diagonal matrix containing the sum of the power spectral densities of the signal sections $\mathbf{x}_q^{\phi}$ along its diagonal, and

$$\mathbf{p}_{xy} = \sum_{q=1}^{Q}\sum_{\phi=1}^{K}\bar{\mathbf{X}}_q^{\phi+}\mathbf{F}_N\mathbf{y}_{d,q}^{\phi} \tag{6.37}$$

is a vector containing the cross-spectral density of the signal sections $\mathbf{x}_q^{\phi}$ and $\mathbf{y}_{d,q}^{\phi}$.

However, if $N = L \neq N_h$, we still need to include the constraints, and the optimized filter is given by

$$\bar{\mathbf{g}} = \mathbf{P}_{xx}^{-1}\mathbf{p}_{xy} - \mathbf{P}_{xx}^{-1}\mathbf{A}\left(\mathbf{A}^{+}\mathbf{P}_{xx}^{-1}\mathbf{A}\right)^{-1}\left(\mathbf{A}^{+}\mathbf{P}_{xx}^{-1}\mathbf{p}_{xy}\right) \tag{6.38}$$

6.2.3 Extending PACF to Higher Dimensions

We can extend PACF to higher dimensions. To illustrate this, we demonstrate how this works for the 2D case.

In 2D, we need to account for the overlap of previous and future sections in both the hori-

zontal and vertical directions. Suppose we are processing section (ϕ_1, ϕ_2). Here, an index of 1 indicates the vertical dimension and an index of 2 indicates the horizontal direction. Then, we define the amount of overlap for surrounding sections in both dimensions. We consider sections above the current section $(\phi_1 - \phi_1^{'})$, sections below the current section $(\phi_1 + \phi_1^{'})$, sections to the left of the current section $(\phi_2 - \phi_2^{'})$, and sections to the right of the current section $(\phi_2 + \phi_2^{'})$. The amount of overlap in each dimension is given by

$$\sigma_1(\phi_1^{'}) = \max(0,\ N_1 - \phi_1^{'} L_1) \tag{6.39}$$

$$\sigma_2(\phi_2^{'}) = \max(0,\ N_2 - \phi_2^{'} L_2) \tag{6.40}$$

where $\phi_1^{'}$ and $\phi_2^{'}$ are positive integers. As in the 1D case, the function for $\sigma_1(\phi_1^{'})$ is the same for sections above and below the current section and the function for $\sigma_2(\phi_2^{'})$ is the same for sections to the left and to the right of the current section (because of symmetry). As before, if $N_1 - \phi_1^{'} L_1$ (or $N_2 - \phi_2^{'} L_2$) is negative, there is no overlap, and the function $\sigma_1(\phi_1^{'})$ (or $\sigma_2(\phi_2^{'})$) returns 0. Next, we find the highest value of $\phi_1^{'}$ $(\phi_2^{'})$ in which section $\phi_1 \pm \phi_1^{'}$ $(\phi_2 \pm \phi_2^{'}$) will overlap section ϕ_1 (ϕ_2),

$$\phi_{1,max}^{'} = \max \phi_1^{'} \qquad | \ \sigma_1(\phi_1^{'}) > 0 \tag{6.41}$$

$$\phi_{2,max}^{'} = \max \phi_2^{'} \qquad | \ \sigma_2(\phi_2^{'}) > 0 \tag{6.42}$$

Note that the *last* $\sigma_1(\phi_1^{'})$ rows of sections $\phi_1 - \phi_1^{'}$ (above the current section) overlap with the *first* $\sigma_1(\phi_1^{'})$ rows of section ϕ_1, and the *first* $\sigma_1(\phi_1^{'})$ rows of sections $\phi_1 + \phi_1^{'}$ (below the current section) overlap with the *last* $\sigma_1(\phi_1^{'})$ rows of section ϕ_1. Similarly, the *last* $\sigma_2(\phi_2^{'})$ columns of sections $\phi_2 - \phi_2^{'}$ (to the left of the current section) overlap with the *first* $\sigma_2(\phi_2^{'})$ columns of section ϕ_2, and the *first* $\sigma_2(\phi_2^{'})$ columns of sections $\phi_2 + \phi_2^{'}$ (to the right of the current section) overlap

with the *last* $\sigma_2(\phi_2^{'})$ columns of section ϕ_2. We illustrate the eight different cases of overlap that can occur in 2D in Fig. 6.8. Note that in cases 1, 3, 6, and 8, the overlap is partial in both dimensions.

Like the 1D case, we must now write expressions for the output of section (ϕ_1, ϕ_2), $\bar{\mathbf{y}}_{a,q}^{\phi_1,\phi_2}$. In this case, each output is a function of the current section (ϕ_1, ϕ_2) and the sections surrounding it. In our notation, $\bar{\mathbf{X}}_q^{a,b}$ represents a diagonal matrix with the vectorized 2D DFT of section (a, b) along the diagonal. We can write the actual output of each section as[5]

$$\begin{aligned}\bar{\mathbf{y}}_{a,q}^{\phi_1,\phi_2} &= \sum_{\phi_1^{'}=-\phi_{1,max}^{'}}^{\phi_{1,max}^{'}} \sum_{\phi_2^{'}=-\phi_{2,max}^{'}}^{\phi_{2,max}^{'}} \mathbf{F}_{N_1\times N_2} \widetilde{\mathbf{R}}_{\phi_1^{'},\phi_2^{'}}^{N_1\times N_2} \mathbf{F}_{N_1\times N_2}^{-1} \bar{\mathbf{X}}_q^{\phi_1+\phi_1^{'},\phi_2+\phi_2^{'}} \bar{\mathbf{g}} \\ &= \mathbf{J}_q^{\phi_1,\phi_2} \bar{\mathbf{g}} \end{aligned} \tag{6.43}$$

where

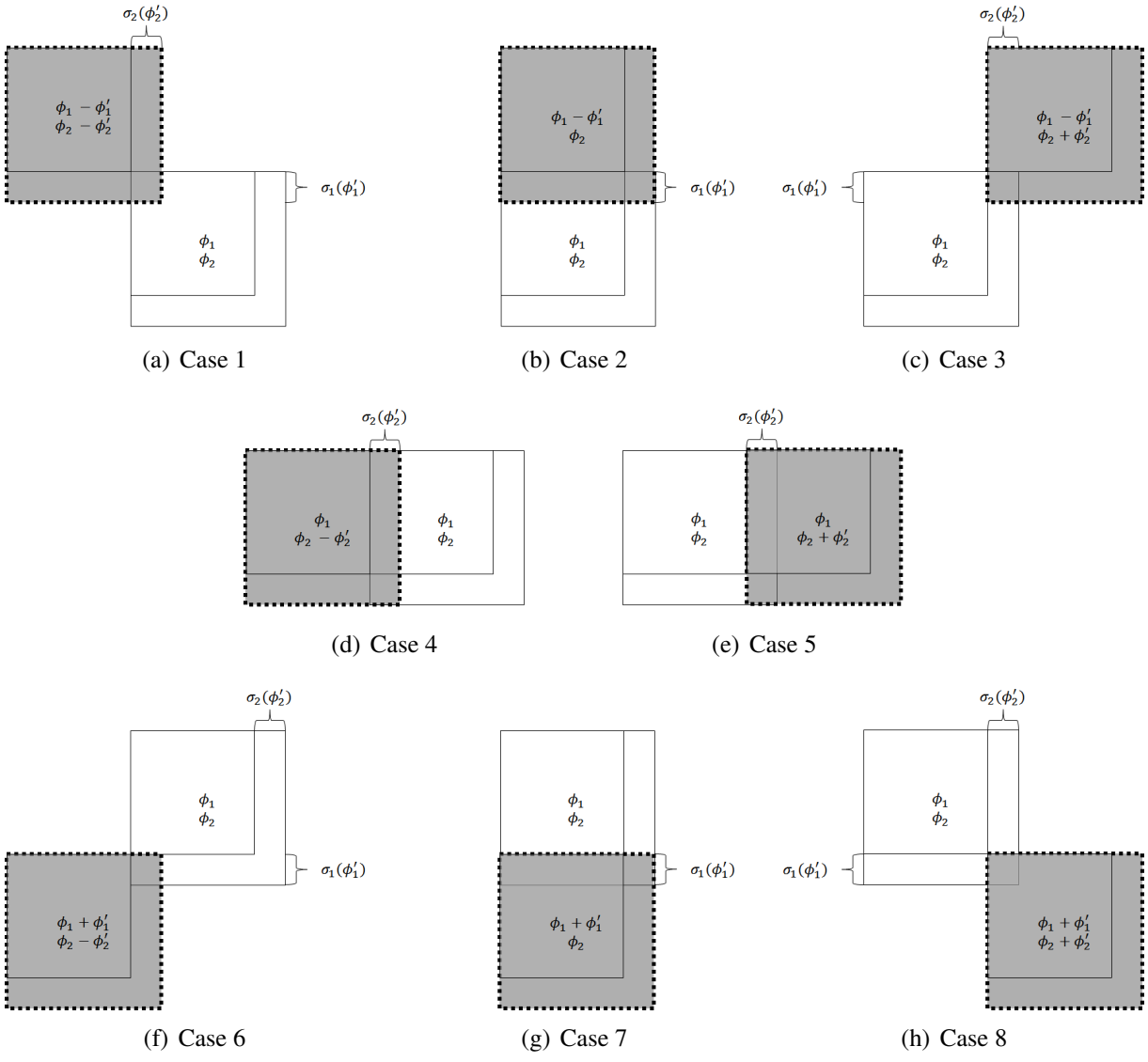
$$\mathbf{J}_q^{\phi_1,\phi_2} = \sum_{\phi_1^{'}=-\phi_{1,max}^{'}}^{\phi_{1,max}^{'}} \sum_{\phi_2^{'}=-\phi_{2,max}^{'}}^{\phi_{2,max}^{'}} \mathbf{F}_{N_1\times N_2} \widetilde{\mathbf{R}}_{\phi_1^{'},\phi_2^{'}}^{N_1\times N_2} \mathbf{F}_{N_1\times N_2}^{-1} \bar{\mathbf{X}}_q^{\phi_1+\phi_1^{'},\phi_2+\phi_2^{'}} \tag{6.44}$$

Note that $\mathbf{F}_{N_1\times N_2}$is an $N_1N_2 \times N_1N_2$ matrix that computes the vectorized 2D DFT of an image that is padded to size $N_1 \times N_2$ and then vectorized. Similarly, $\mathbf{F}_{N_1\times N_2}^{-1}$ is an $N_1N_2 \times N_1N_2$ matrix that computes the vectorized 2D IDFT of a vectorized 2D DFT. For clarity, note that

$$\bar{\mathbf{g}} = \mathbf{F}_{N_1\times N_2} \mathbf{g} \tag{6.45}$$

$$\mathbf{g} = \mathbf{F}_{N_1\times N_2}^{-1} \bar{\mathbf{g}} \tag{6.46}$$

[5]Note that in this equation, we have abused notation slightly. While we have defined $\phi_1^{'}$ and $\phi_2^{'}$ as positive integers, we include the sign in their value here to more succinctly express sections to the left of or above the current section.



(a) Case 1 (b) Case 2 (c) Case 3

(d) Case 4 (e) Case 5

(f) Case 6 (g) Case 7 (h) Case 8

Figure 6.8: Eight different cases for which sections can overlap in 2D PAOLA.

Using previous notation, $\mathbf{F}^{-1}_{N_1 \times N_2} = \mathbf{\Omega\Phi}$, where $\mathbf{\Omega}$ and $\mathbf{\Phi}$ are given by Eq. 3.15 and 3.12. Also note that

$$\mathbf{F}_{N_1 \times N_2} = N_1 N_2 (\mathbf{F}^{-1}_{N_1 \times N_2})^* \tag{6.47}$$

Finally, we define the 2D selection matrix $\widetilde{\mathbf{R}}^{N_1 \times N_2}_{\phi_1^{'},\phi_2^{'}}$. This is an $N_1 N_2 \times N_1 N_2$ matrix that selects a portion of an adjacent section and shifts it so that it overlaps appropriately with the current section. First, when $\phi_1^{'} = \phi_2^{'} = 0$, $\widetilde{\mathbf{R}}^{N_1 \times N_2}_{\phi_1^{'},\phi_2^{'}}$ is equal to the $N_1 N_2 \times N_1 N_2$ identity matrix ($\mathbf{I}_{N_1 N_2}$), and the product $\mathbf{F}_{N_1 \times N_2} \widetilde{\mathbf{R}}^{N_1 \times N_2}_{0,0} \mathbf{F}^{-1}_{N_1 \times N_2} \bar{\mathbf{X}}^{\phi_1,\phi_2}_q \bar{\mathbf{g}}$ simplifies to $\bar{\mathbf{X}}^{\phi_1,\phi_2}_q \bar{\mathbf{g}}$, which is the output correlation of the current section. For surrounding sections (cases other than $\phi_1^{'} = \phi_2^{'} = 0$), the definition of $\widetilde{\mathbf{R}}^{N_1 \times N_2}_{\phi_1^{'},\phi_2^{'}}$ is significantly more complex, and is presented in Appendix E, along with examples of how this selection matrix works.

At this point, the problem follows the same path as that in the 1D case. In this case, we minimize

$$\min_{\bar{\mathbf{g}}} \sum_{q=1}^{Q} \sum_{\phi_1=1}^{K_1} \sum_{\phi_2=1}^{K_2} (\mathbf{J}^{\phi_1,\phi_2}_q \bar{\mathbf{g}} - \bar{\mathbf{y}}^{\phi_1,\phi_2}_{d,q})^+ (\mathbf{J}^{\phi_1,\phi_2}_q \bar{\mathbf{g}} - \bar{\mathbf{y}}^{\phi_1,\phi_2}_{d,q}) - \boldsymbol{\omega}^+ \mathbf{A}^+ \bar{\mathbf{g}} \tag{6.48}$$

where $\mathbf{A}$ is formed in a similar fashion for 2D constraints as outlined in Section 3.2. Note that these constraints, however, may be slightly different from the zero-aliasing constraints if we are building a template that is smaller than the object of interest. After solving this optimization problem, the 2D PACF is given by the expression in Eq. 6.31 for overlap cases with constraints, Eq. 6.34 for overlap cases with no constraints ($N_1 = N_{h,1}$ and $N_2 = N_{h,2}$), Eq. 6.35 for when there is no overlap and no constraints ($N_1 = L_1 = N_{h,1}$ and $N_2 = L_2 = N_{h,2}$), and Eq. 6.38 for no overlap cases with constraints ($N_1 = L_1 \neq N_{h,1}$ and $N_2 = L_2 \neq N_{h,2}$). In these expressions, $\mathbf{P}_{jj}$, $\mathbf{p}_{jy}$, $\mathbf{P}_{xx}$, and $\mathbf{p}_{xy}$ are given by

$$\mathbf{P}_{jj} = \sum_{q=1}^{Q} \sum_{\phi_1=1}^{K_1} \sum_{\phi_2=1}^{K_2} \mathbf{J}_q^{\phi_1,\phi_2+} \mathbf{J}_q^{\phi_1,\phi_2} \tag{6.49}$$

$$\mathbf{p}_{jy} = \sum_{q=1}^{Q} \sum_{\phi_1=1}^{K_1} \sum_{\phi_2=1}^{K_2} \mathbf{J}_q^{\phi_1,\phi_2+} \bar{\mathbf{y}}_{d,q}^{\phi_1,\phi_2} \tag{6.50}$$

$$\mathbf{P}_{xx} = \sum_{q=1}^{Q} \sum_{\phi_1=1}^{K_1} \sum_{\phi_2=1}^{K_2} \bar{\mathbf{X}}_q^{\phi_1,\phi_2+} \bar{\mathbf{X}}_q^{\phi_1,\phi_2} \tag{6.51}$$

$$\mathbf{p}_{xy} = \sum_{q=1}^{Q} \sum_{\phi_1=1}^{K_1} \sum_{\phi_2=1}^{K_2} \bar{\mathbf{X}}_q^{\phi_1,\phi_2+} \mathbf{F}_{N_1 \times N_2} \mathbf{y}_{d,q}^{\phi} \tag{6.52}$$

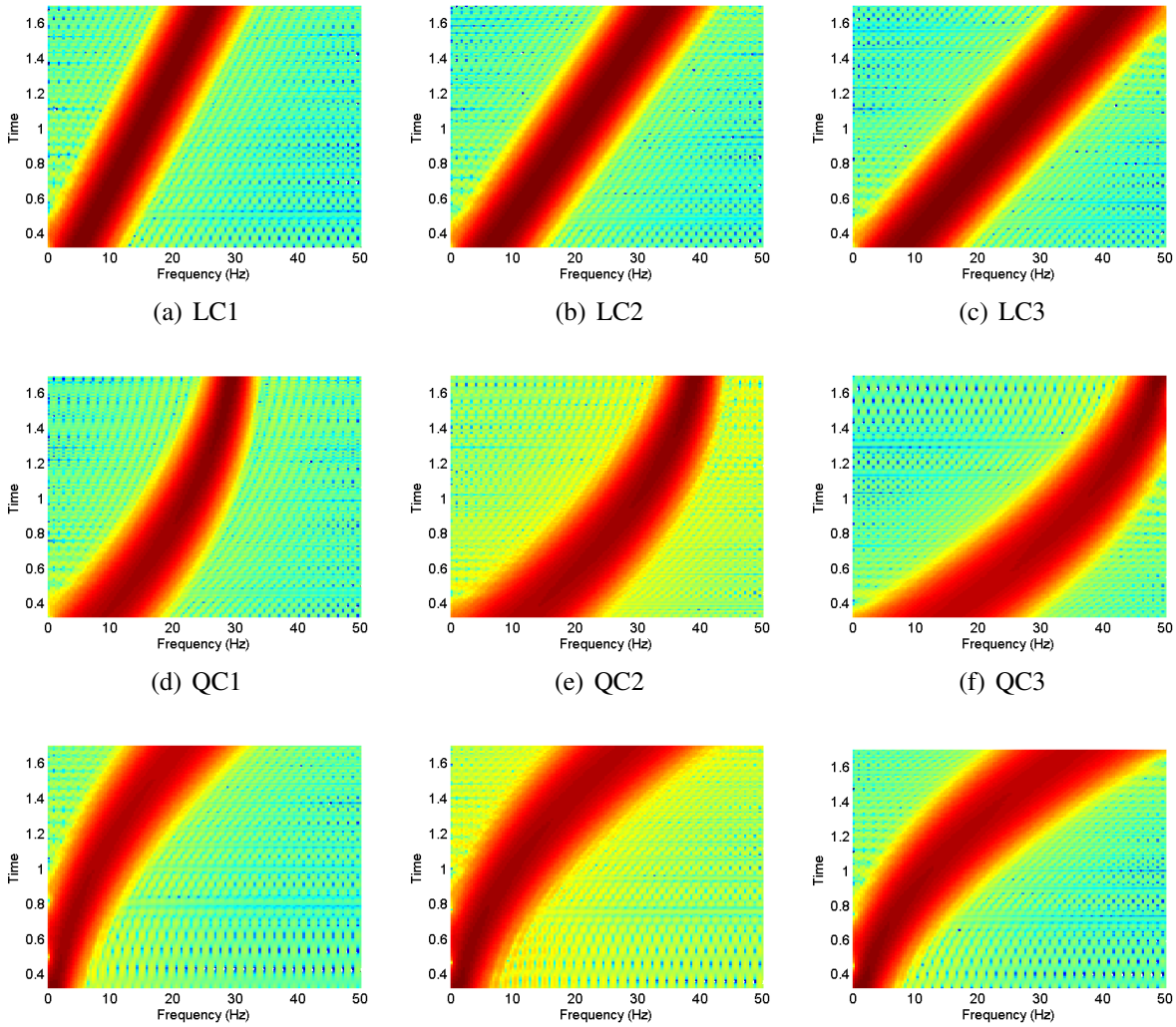
6.3 Experiments

To demonstrate PACFs, we conduct a variety of experiments on 1D and 2D synthetic data.

6.3.1 1D Chirp Detection

First, we demonstrate PACFs in 1D. We generate a 1D dataset of chirp signals. We consider nine chirp signals:

- LC1: Linear chirp, 0 to 30 Hz;
- LC2: Linear chirp, 0 to 40 Hz;
- LC3: Linear chirp, 0 to 50 Hz;
- QC1: Quadratic chirp, 0 to 30 Hz;
- QC2: Quadratic chirp, 0 to 40 Hz;
- QC3: Quadratic chirp, 0 to 50 Hz;
- CC1: Concave chirp, 0 to 30 Hz;
- CC2: Concave chirp, 0 to 40 Hz;
- CC3: Concave chirp, 0 to 50 Hz.



(a) LC1 (b) LC2 (c) LC3

(d) QC1 (e) QC2 (f) QC3

(g) CC1 (h) CC2 (i) CC3

Figure 6.9: Spectrograms of chirps in the nine class chirp dataset.

These chirp signals are all sampled at a sampling rate of $F_s = 100$ Hz and are 2.01 seconds in duration ($N_s = 201$ samples each). Spectrograms of each chirp are shown in Fig. 6.9.

For each class, we train the PACF using $Q = 10$ training signals $\mathbf{x}_q$ of length $N_x = 1000$ (10 seconds). Each of the training signals contains a single, randomly delayed chirp (such that the chirp is fully contained within the N_x samples) with additive white Gaussian noise (AWGN) of zero mean and variance 0.0225 (signal to noise ratio, SNR ~ 13.7 dB). We train the MOSSE filter from a second set of aligned training signals $\mathbf{s}_q$ (of length N_s) which are the cropped and

aligned chirps extracted from the signals $\mathbf{x}_q$.

We generate $P = 500$ testing signals (of length N_x) for each class (4500 test signals total) in a similar fashion to the training set, except with AWGN of variance 0.225 (SNR ~ 3.7 dB). We then apply the CFs to the test signals using the PAOLA architecture. We sweep the DFT size and fix the cut length at $L = N_s$. For the PACF and MOSSE filters[6] we use a template length $N_h = N_s$. However, the power of the PACF formulation includes the ability to change the template size, i.e. we can solve for a template such that $N_h < N_s$. Therefore, we try two methods. First, we set $N_h = N + 1 - N_s$, denoted PACF v1; second, we set $N_h = \min(N_s, 2(N + 1 - N_s))$, denoted PACF v2. PACF v1 corresponds to no aliasing, but the template is considerably smaller than the size of the chirps for small values of N. PACF v2 allows some aliasing by using a template that is approximately two times as long as PACF v1.

We generate ROC curves for each class by varying the threshold for detection. For each threshold, a correct detection must be of the correct class and location (within ± 5 samples of a ground truth). Otherwise, it is a false alarm. To obtain a single score for the recognition performance, we integrate the ROC curve from 0 to 1 false alarms per second (FA/s), denoting this quantity as P_1. We repeat the above experiment 20 times and then average the results over all experiments and filters. Our results are shown in Fig. 6.10, which also shows the time (experimental) needed to compute a correlation at a given N using PAOLA.

Note that the PACF method outperforms the MOSSE filter when aliasing due to circular correlation is significant (lower values of N). As N increases (and aliasing decreases), the two methods converge to perfect performance in terms of chirp detection. Also note that, for small values of N, PACF v1 and PACF v2 perform poorly (due to using a very small template size). However, as the template size gets larger (with increasing N), performance exceeds that of the standard PACF method that uses $N_h = N_s$.

Note that PACF, PACF v1, and PACF v2 offer three examples of the design space of PACFs. One one end of the spectrum, we have PACF, which uses a large template size (equal in size to the

[6]The MOSSE does not use any constraints and is simply cropped to this length.

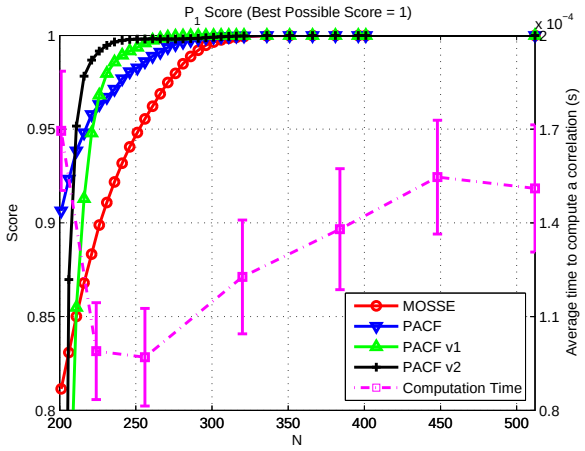
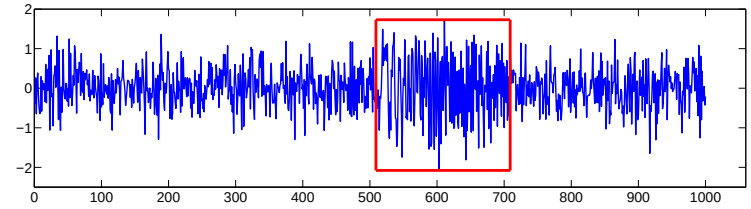


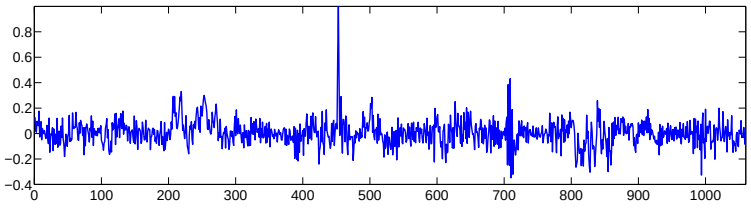
Figure 6.10: Performance score on the 1D chirp dataset, for the PACF and MOSSE filters, as a function of N, the DFT size. Note that the best possible score is 1. The timing curve shows the 95% confidence intervals of our computational evaluation.

training chirps, $N_h = N_s$) and features the most aliasing due to circular correlation. On the other end of the spectrum, we have PACF v1, which uses a smaller template size which forces circular correlation to zero. In the middle, we have PACF v2, which allows some aliasing by choosing a template size between PACF v1 and PACF. Although we have not proven that PACF v2 uses the best possible choice of parameters, we have shown that this method, which is a balance between PACF and PACF v1, has led to superior detection performance for most DFT sizes. Furthermore, to the best of our knowledge, this is the first time that a CF has been trained such that is shorter than the training signal size. This type of template design was never possible in the frequency domain until we introduced zero-aliasing constraints in [41].

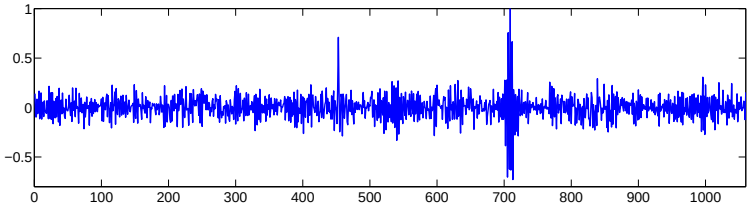
This example also illustrates a key benefit of PACFs. Assuming L is fixed, we could use $N = 401$ to completely avoid aliasing. This is, however, a prime number and inefficient; therefore, we would typically choose a larger N (likely $N = 512$). With PACFs, however, we could use $N = 256$ and achieve a recognition score of $P_1 = 0.9981$ using PACF v2. According to our experiments, this would be 1.56 times faster than using a $N = 512$, and would be more memory efficient as well. These savings are even more important for 2D and 3D applications. This underscores the primary motivation of PACFs - to reduce the DFT size required to compute correlations.



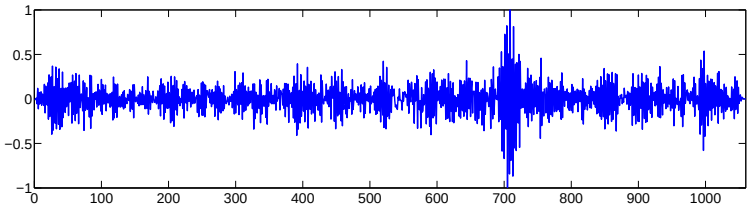
(a) Input Signal, Class = QC2



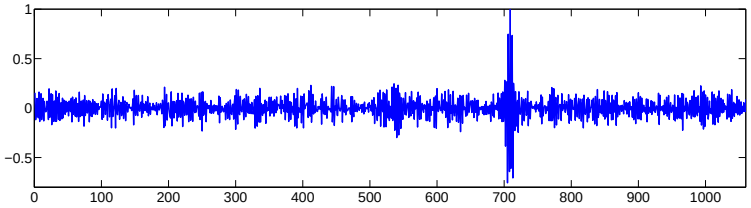
(b) Output Correlation, MOSSE - QC2



(c) Output Correlation, PACF - QC2



(d) Output Correlation, PACF v1 - QC2



(e) Output Correlation, PACF v2 - QC2

Figure 6.11: Example performance on a test signal using PAOLA ($N = 256$).

We show correlation outputs on test signals in Fig. 6.11. In this case, we show an input test signal with chirp QC2 (quadratic chirp, 0-40 Hz). The red box denotes the location of the chirp. We show normalized correlation outputs for MOSSE, PACF, PACF v1, and PACF v2. Note that the desired output is a peak at $t = 709$. The MOSSE output features severe aliasing effects, including a larger (aliased) peak at $t = 453$. The PACF reduces these effects, but PACF v1 and PACF v2 (which use $N_h = 56$ and $N_h = 112$, respectively) nearly eliminate them, leading to the best performance.

6.3.2 2D Shape Detection

In this experiment, we verify the performance of PACF in 2D. We generate ten different shapes which serve as our classes, and show these in Fig. 6.12. These shapes are all $N_{s,1} \times N_{s,2} = 16 \times 16$ pixels. We choose this smaller size for computational reasons (the matrix $\mathbf{P}_{jj}$ is of size $N_1 N_2 \times N_1 N_2$).

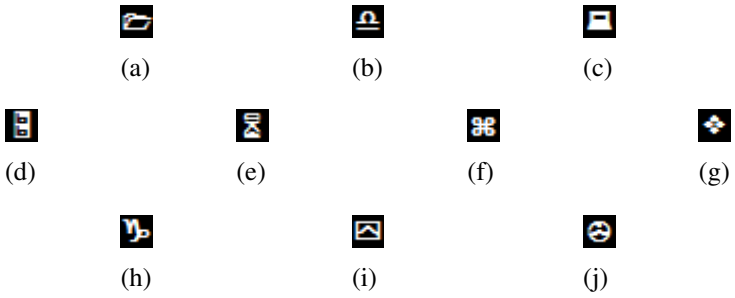
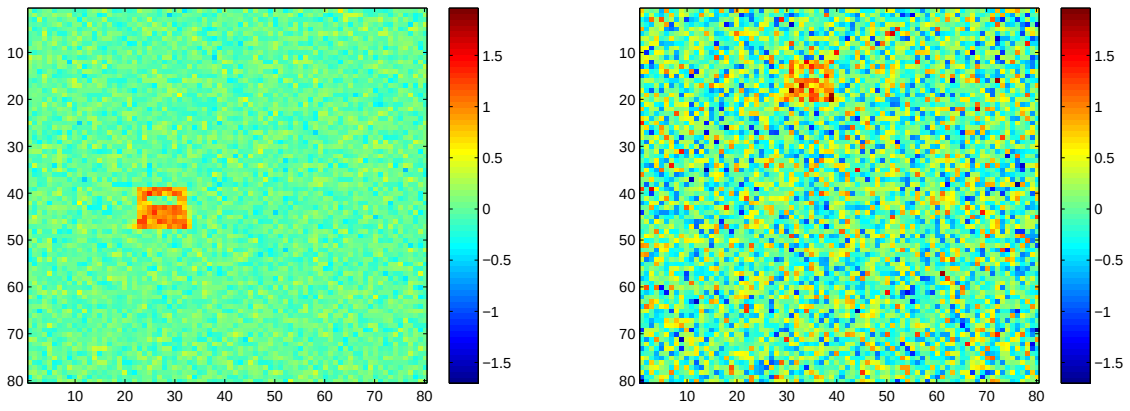


Figure 6.12: Classes for 2D PACF verification experiments.

We model the 2D experiment after the 1D chirp experiment. Here, we build $Q = 10$ training images $\mathbf{x}_q$ per class using $N_{x,1} \times N_{x,2} = 80 \times 80$. We randomly shift the target such that it is completely contained within each 80×80 pixel image. We then use AWGN of zero mean and variance 0.0225. We train the PACF using $\mathbf{x}_q$ and the MOSSE filter using a second set of aligned training images $\mathbf{s}_q$ (of size $N_{s,1} \times N_{s,2}$) which are the cropped and aligned symbols extracted from $\mathbf{x}_q$.

For testing the filters, we set the PAOLA section size at $L_1 \times L_2 = 16 \times 16$ and we sweep the DFT size $N_1 \times N_2$ from 16×16 to 32×32. We generate $P = 150$ test images per class and use AWGN of zero mean and variance 0.225. For comparison, we show a training and testing image in Fig. 6.13.



(a) Training Image (b) Test Image

Figure 6.13: Training and testing image examples for the 2D PACF example.

As in the 1D case we use a template size of $N_{h,1} \times N_{h,2} = N_{s,1} \times N_{s,2}$ for the MOSSE and PACF filters. However, the power of the PACF formulation includes the ability to change the template size. Once again, we implement PACF v1 ($N_{h,1} \times N_{h,2} = N_1 + 1 - N_{s,1} \times N_2 + 1 - N_{s,2}$) and PACF v2 ($N_{h,1} \times N_{h,2} = \min(N_{s,1}, 2(N_1 + 1 - N_{s,1})) \times \min(N_{s,2}, 2(N_2 + 1 - N_{s,2}))$). Recall that PACF v2 allows some aliasing but PACF v1 allows no aliasing.

We generate ROC curves for each class by varying the threshold for detection. For each threshold, a correct detection must be of the correct class and location (within ± 5 samples in each dimension of a ground truth). Otherwise, it is a false alarm. To obtain a single score for the recognition performance, we integrate the ROC curve from 0 to 1 false alarms per image (FA/i), denoting this quantity as P_1. We repeat the above experiment 8 times and then average the results over all experiments and filters. Our results are shown in Fig. 6.14.

In the 2D case, we note that the PACF once again outperforms the MOSSE filter. However,

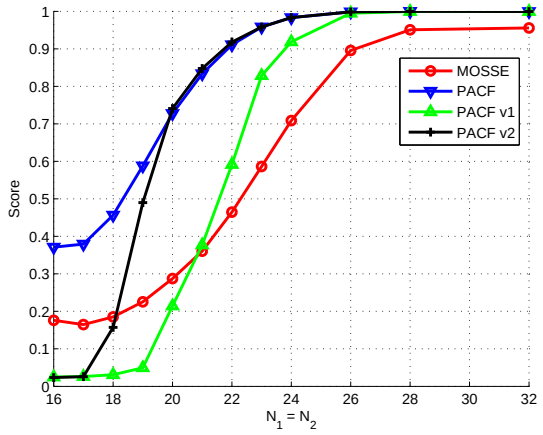


Figure 6.14: Performance score on the 2D shape dataset, for the PACF and MOSSE filters, as a function of N, the DFT size. Note that the best possible score is 1.

we find that PACF v1 in the case does not perform well, which indicates that the advantage of training templates smaller than the target in 2D is diminished compared to the 1D case. For PACF v2, we achieve slightly better performance than PACF for DFT sizes greater than or equal to 20×20. However, there is not a noticeable improvement like what was observed in the 1D case.

We next turn our attention to a few variations of PACF. First, we note that training a PACF in 2D is computationally complex. Most of the complexity is in forming matrix $\mathbf{P}_{jj}$. This matrix is a function of $\mathbf{J}_q^{\phi_1,\phi_2}$ (Eq. 6.44), which is computed for every section in the training image. For this experiment, this is 25 sections per training image. Because iterating over every section is time consuming, we instead train a PACF using only sections that contain the target. This is usually four sections per training image. Therefore, we can greatly reduce the training time. To see the effect this has on the filter performance, we repeat the previous experiment 6 times and average the results for the MOSSE, PACF, and PACF lite, where PACF lite denotes the reduced training PACF. The results are shown in Fig. 6.15. Note that by removing sections that do not contain the target, we reduce performance slightly, but PACF lite still outperforms MOSSE. We speculate that training over the entire image adds some noise tolerance to the PACF filter, which accounts for the gap between PACF and PACF lite. Indeed, we generally noticed that PACF

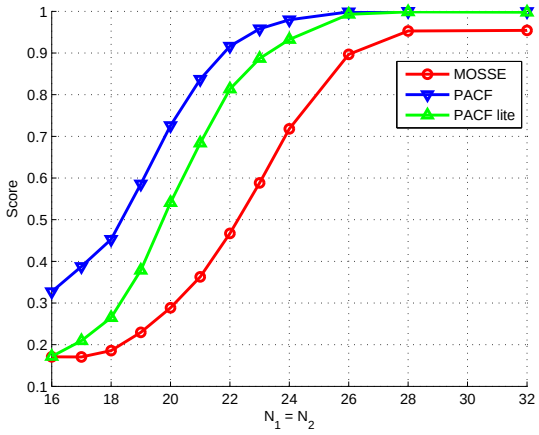


Figure 6.15: Performance score on the 2D shape dataset, for the PACF and MOSSE filters, as a function of N, the DFT size. Note that the best possible score is 1.

exhibited broader peaks than PACF lite, which is typically true of filters that are designed to be more noise tolerant.

We investigate the effect of training set size for both MOSSE and PACF. We repeat the previous experiment 6 times, except this time we use $Q = 10$, $Q = 35$, and $Q = 100$ training images. We present the results in Fig. 6.16. Note that the biggest jump in performance is found going from 10 to 35 training images. PACF especially benefits from more training images for smaller values of N, when aliasing is greatest.

Finally, we present example correlation outputs for both MOSSE and PACF in Fig. 6.17. These outputs illustrate several ways in which PACF leads to better performance. First, in Fig. 6.17a and b, the target is located at $y = 62$, $x = 73$. In both cases, the highest peak is located at the correct location. In the MOSSE case, there is significant aliasing present, including an aliased peak. In the PACF case, however, the aliasing is greatly reduced, leading to a single peak. Note that this is one reason why PACF outperforms MOSSE when using common metrics such as PCE or PSR. In Fig. 6.17c and d, the the target is located at $y = 58$, $x = 57$. In this case, the MOSSE filter exhibits severe aliasing, and the highest peak is not at the correct location. The PACF, on the other hand, features a peak in the correct location and low sidelobes around it. Finally, in Fig. 6.17e and f, we show a situation in which PACF is unable to completely suppress

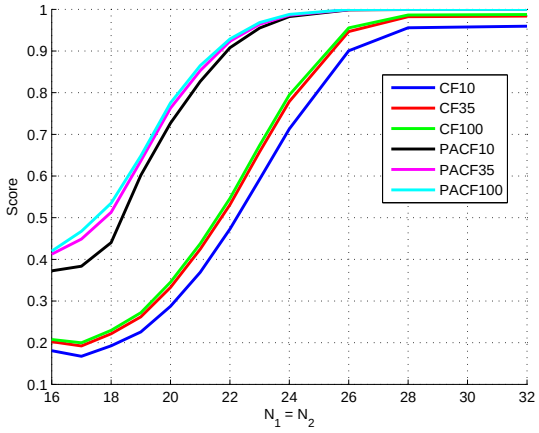


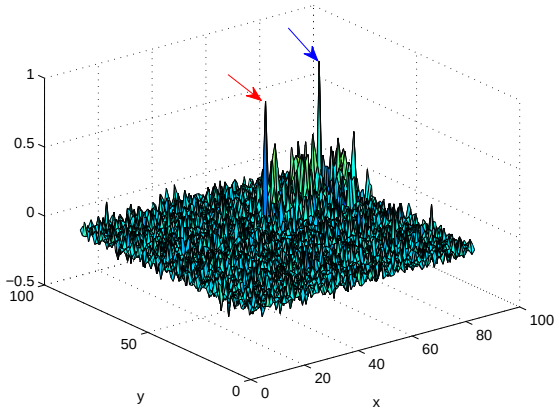
Figure 6.16: Performance score on the 2D shape dataset, for the PACF and MOSSE filters, as a function of N, the DFT size. Note that the best possible score is 1.

the aliasing. In this example, the target is located at $y = 72$, $x = 65$. In the MOSSE case, the aliased peak is larger than the authentic peak. However, for the PACF case, the correct peak is larger than the aliased peak. Therefore, we see that PACF can outperform MOSSE even if it does not completely eliminate aliasing.

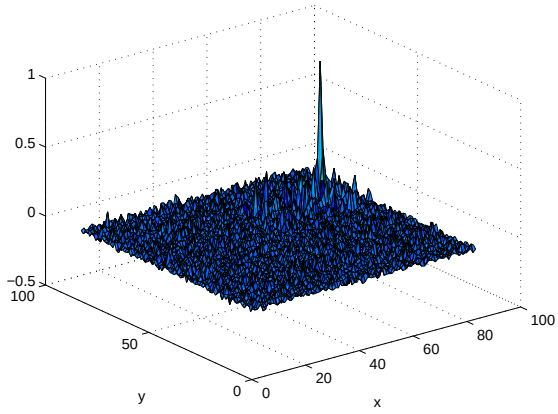
6.4 Conclusions

In this chapter, we have demonstrated, for the first time, a method to train CFs for a block filtering architecture that intentionally allows aliasing. This signifies a departure from the conventional way of thinking about how CFs should be trained. We have shown that CFs trained in this manner can still perform well, despite operating in the presence of aliasing. We summarize below the contributions in this chapter.

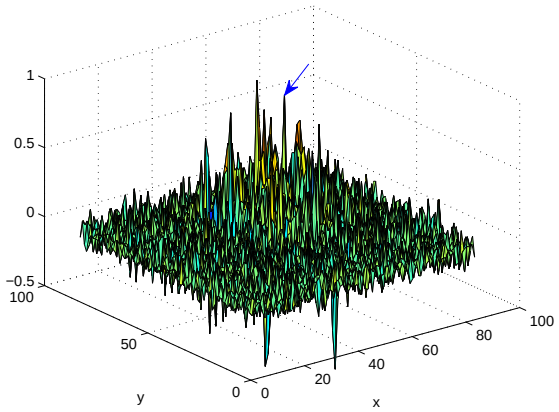
- We presented PAOLA, which is a generalized version of OLA.
- We showed how PAOLA uses less memory and requires less computations than OLA, at the expense of introducing aliasing into the output signal.
- We presented analytical expressions for computation and memory requirements for PAOLA.



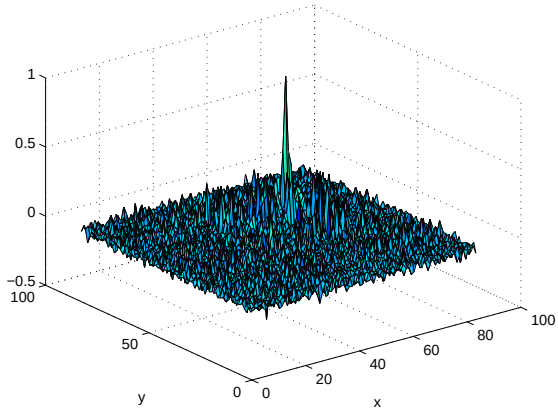
(a) MOSSE



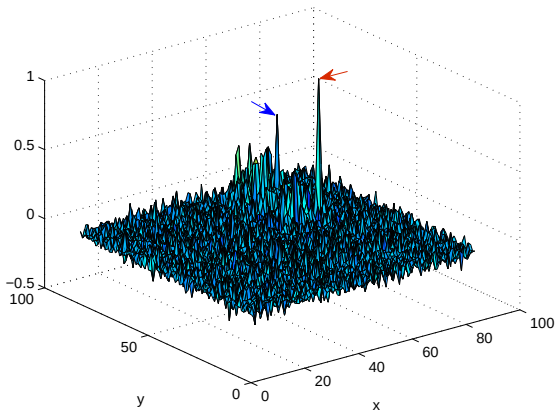
(b) PACF



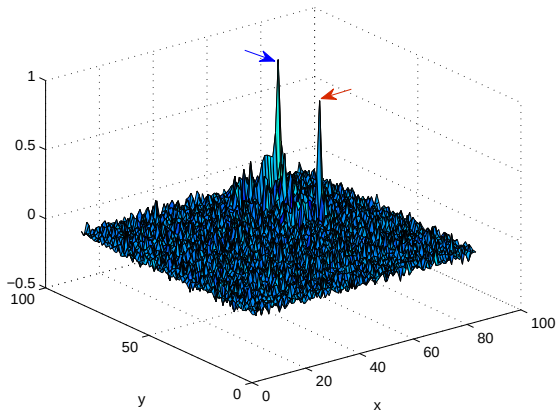
(c) MOSSE



(d) PACF



(e) MOSSE



(f) PACF

Figure 6.17: Example correlation peaks for MOSSE and PACF. In all cases, a blue arrow indicates the correct peak, whereas a red arrow indicates an incorrect peak.

- We have introduced PACFs, which offer a method to train CFs while taking into account the aliasing that results from the PAOLA architecture.
- We have extended the PACF formulation to 2D.
- We have shown better performance using PACF compared to a baseline CF when detecting chirp signals in 1D.
- We have illustrated that it is possible to train CF templates that are shorter than the training signals to reduce aliasing. These templates can achieve good recognition performance. To the best of our knowledge, our method is the first ever to train a CF template that is smaller than the training signals. This is made possible by zero-aliasing constraints.
- We have shown better performance using PACF compared to a baseline CF when detecting shapes in 2D.
- We have discussed the effects of train set size and training methods on PACF performance.

The ideas in this chapter lie at the crossroads of several ideas presented in this thesis. In the previous chapter, we focused on reducing the computation and memory required to apply a CF to a signal. In this chapter, we take this idea to the next step by developing a new training method that intentionally allows aliasing in order to achieve better computational and memory performance. By leveraging ideas from ZACFs, we developed PACFs, which perform well in the presence of aliasing. In the next chapter, we summarize the contributions of this thesis and present directions for future research.

Chapter 7

Conclusions and Future Work

In this thesis, we introduced several new designs for CFs that, for the first time, take into account the aliasing effects caused by the DFT. First, we focused on the aliasing effects that are a result of using DFTs to compute the CF. The element-wise multiplication of two DFTs in the frequency domain corresponds to a circular correlation in the time/space domain. In previous CF research, this circular correlation was assumed to be roughly equivalent to a linear correlation. In this thesis, we presented a solution to this long-standing problem by introducing zero-aliasing CFs (ZACFs), which offer a design that accurately reflects a linear correlation. We have shown that ZACFs outperform traditional CFs on a variety of datasets. In addition, we present several alternatives to ZACFs that are more computationally tractable.

Next, we investigated the computational and memory requirements necessary for applying CFs to multidimensional data. By conducting a thorough analysis of the overlap-add (OLA) and overlap-save (OLS) algorithms, we have shown that OLA offers a significant computational advantage, whereas OLS offers a significant memory advantage. This analysis prompted us to develop partial-aliasing OLA (PAOLA), which outperforms OLA in terms of computation and memory, at the expense of introducing aliasing into the correlation output. We introduced partial-aliasing CFs (PACFs), which are designed to explicitly minimize this intentional aliasing and produce sharp peaks in situations in which conventional CFs fail.

In this chapter we summarize the main contributions of this thesis. We then draw key conclusions and outline possible directions for future work.

7.1 Contributions

- **Aliasing issues in CF designs**: We illustrated the aliasing issues that are present in conventional CF formulations. We did this in two different ways. First, we showed that using different DFT sizes while designing CFs leads to different templates. These templates perform best when the DFT size at training is the same as the DFT size at testing. However, if the DFT size changes, performance suffers considerably. Second, we showed that the representation of correlation itself in conventional CF designs is in fact a circular correlation. Because CFs are applied to test data using linear correlation, the conventional CF formulation is inconsistent with the original intention to use CFs under the paradigm of linear correlation. Because of this, designed CFs are not optimal. Furthermore, we debunked a common assumption that padding training images with more zeros solves this circular correlation issue.
- **Introduction of ZACFs**: We introduced a new class of CFs, known as ZACFs, that introduce zero-aliasing constraints into the CF design methodology. These constraints force the tail of the CF template to zero, which means that the element-wise multiplication of the CF and DFTs of the padded training images corresponds to a linear correlation. This step makes the CF design consistent with how it is applied to test data. In addition, the ZACF yields a CF template that better satisfies the original optimization criteria, which we demonstrated explicitly with the MACE filter.
- **Extension of ZACFs to popular filter designs**: We extended the ZACF framework to many existing constrained, unconstrained, and inequality constrained CF designs. In doing this, we produced closed-form solutions for the zero-aliasing versions of many popular CF

designs.

- **Alternatives to ZACFs**: Acknowledging that computing the closed-form ZACFs can be computationally intensive, we have introduced several different alternatives to ZACFs. First, the reduced-aliasing CF (RACF) leverages a smaller DFT size in tandem with zero-aliasing constraints to reduce aliasing but not completely eliminate it. We have also developed an alternate CF design method, known as tail-energy minimization (TEM), that minimizes the energy of the CF template's tail, rather than constraining it to zero as the ZACF does. The benefit of this method is that it does not require reformulation of existing CF designs. We also developed reduced-aliasing TEM (RATEM), which mirrors RACF in the same way that TEM mirrors ZACF. Finally, we investigated an accelerated proximal gradient descent (APGD) method to numerically compute ZACF templates.
- **Numerical results for ZACFs**: We showed numerically that ZACFs outperform conventional CFs for a variety of applications, including face recognition, automatic target recognition, and eye localization. We have also presented results comparing both the speed and recognition performance of RACF, TEM, RATEM, and APGD, which illustrates that these methods are suitable replacements for ZACFs. APGD is our preferred method, as it is very fast and memory efficient.
- **Comparison of OLA and OLS**: We performed a detailed analysis of the OLA and OLS filtering algorithms for multidimensional applications. We presented theoretical expressions for computational complexity which we validated experimentally. In performing this analysis, we showed that OLA offers a computational advantage over OLS, which is amplified in higher dimensions. We also showed how parameter selection may be used to minimize the computational complexity for each algorithm. While OLA outperforms OLS computationally, we showed that OLS offers a distinct memory advantage over OLA. We characterized this by developing theoretical expressions for the amount of memory needed for each algorithm. We performed this memory analysis for both 2D and 3D and offer in-

sight into various design considerations such as scanning direction and various high-level memory management techniques.

- **Development and analysis of PAOLA**: We illustrated a third block filtering technique, known as PAOLA, which is based on OLA but offers reduced computational and memory requirements. The drawback is that this scheme introduces aliasing due to circular correlation in the output correlations. Acknowledging this, we analyzed the computational and memory requirements of PAOLA.
- **Introduction of PACFs**: Using the PAOLA filtering architecture, we introduced a new class of CFs known as PACFs. Unlike conventional CFs, the PACF design formulation takes into account the details of the filtering architecture, including the section size and the DFT size. It then explicitly minimizes the aliasing effects due to circular correlation by taking into account the DFT size and the amount of overlap between output sections. We have presented PACF formulations for both 1D and 2D applications, and have demonstrated that PACFs outperform conventional CFs when aliasing is present for several different applications.
- **CF template size**: Using PACFs, we showed, for the first time, that it is possible to train CF templates that are smaller than the target. This was done to reduce the amount of aliasing present in the correlation outputs. However, the ability to control the size of the CF template is a significant contribution to CF theory and may be used in different ways in the future.

7.2 Conclusions

The work presented in this thesis for the first time focuses on the aliasing effects present in both the design and application of CFs. For many years, CF designs did not account for the aliasing effects caused by the DFT. We have shown that these aliasing effects are significant,

and cannot be ignored. We have eliminated aliasing during training through the ZACF. We have taken advantage of aliasing during the application of CFs through the PACF. Together, the ZACF and PACF formulations offer new and exciting insights to CF theory.

All applications that use CFs can benefit tremendously by adopting zero-aliasing constraints. While we have not tried ZACFs for every application there is, we have seen strong evidence that performance almost always is improved by using ZACFs instead of CFs. This means that there are a wealth of applications that can benefit from the contributions of this thesis.

The computational complexity of applying CFs is often an overlooked topic in the field of CFs. Typically, correlation is performed in the frequency domain using what we call the "conventional" technique. This thesis offers important insight to improving computation and memory requirements for the application of CFs, to a level of detail not found in other works. This information is an important contribution to the field of CFs, and should be invaluable to anyone who wishes to implement correlation or convolution as fast as possible for a given template and test signal.

Our development of PACFs opens the door to an entirely new class of CFs that are designed to work in the presence of aliasing. Throughout all of CF theory, the assumption was always that the CFs were applied to test data in a very specific manner, namely that linear correlation was always used. If we intentionally alias the output using DFTs of inadequate sizes, we create a "problem" under the conventional way of thinking about CFs. However, PACFs show how we can explicitly take this aliasing into account and design CFs that work *with* aliasing. We have only shown the tip of the iceberg; there are certainly many more ideas that may spring forth from the ideas behind PACF.

7.3 Future Directions

There are many exciting ideas that may come from this PhD thesis. Here, we discuss some of these ideas.

Future Work for ZACFs

- In this thesis, we developed a proximal gradient numerical method for computing constrained and unconstrained CFs. This method, in general, is preferred to the closed-form techniques, which require a large amount of memory due to the formation of large matrices. Therefore, one possible area for future work is to extend this proximal gradient technique to more advanced CFs, such as the MMCF, which uses inequality peak constraints instead of equality peak constraints (constrained CFs) or no peak constraints (unconstrained CFs).
- We have presented a number of computational improvements for ZACF in this thesis. However, more work should be done to further speed up the training time for these CFs. For instance, there may be more efficient ways to compute TEM CFs by taking advantage of the structure of the matrices in the formulation.
- In this thesis, we have not discussed any methods by which to train ZACFs iteratively. For example, if we train a ZACF with Q training images, we may want to later update the resulting template with additional training images as they become available. The question here is the development of a method to update the existing ZACF template efficiently.
- We have extended ZACFs to many different types of CFs. However, there are more CFs that our ideas may be applied to. Vector correlation filters [35] or biometric key binding methods [28] are two examples.
- The ideas in ZACFs may also be used to eliminate aliasing in other frequency-domain filtering techniques, such as convolutional sparse coding [40]. In that work, the authors acknowledge aliasing due to circular convolution, but rely on heuristics to attempt to reduce this aliasing. Our zero-aliasing methods can potentially be useful in that work.
- One direct result of our work on ZACFs is the masked correlation filter (MCF). We have developed this as a new method of training CFs based on ZACFs. While it has not been included in this thesis, we feel that this development can revolutionize CFs for recognition. The idea is that a CF of any shape can be trained by leveraging a modified form of zero-

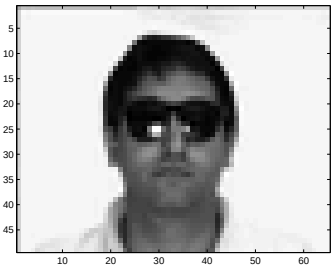
aliasing constraints. Instead of simply constraining the tail of the CF template to zero, we can constrain various regions (in addition to the tail) to zero. This means that a CF of any shape may be built. The formulation of this problem is quite straightforward, and only involves changing the selection matrix $\mathbf{S}$ given in Eq. 3.17. We illustrate this using two examples. First, we train a CF template to ignore a particular portion of the training images in Fig. 7.1. This example uses images from the AR Face Dataset [64]. Next, we demonstrate how a non-rectangular CF may be trained using MCF in Fig. 7.2. In this case, the template is shaped like a "T" to capture the subject's ocular region in addition to the nose and mouth. This example uses images from the FRGC dataset [50].

Future Work for OLA and OLS

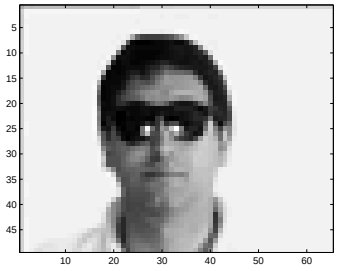
- In this thesis, we have focused on a high-level analysis of the computation and memory required for the OLA and OLS algorithms. We have found that these expressions translate well to experimental results using MATLAB code. However, this work can be extended by developing an optimized code package written in C (or a similar programming language) as a package for researchers to take advantage of this work to speed up correlations and convolutions in everyday signal processing work.
- In practice, the speed and memory requirements of OLA and OLS will be highly dependent on the hardware configuration used to compute correlations. While this is not our area of expertise, both algorithms may be highly optimized given a particular hardware configuration, e.g. different CPUs or GPUs. In addition, FPGA implementation may be considered.

Future Work for PACFs

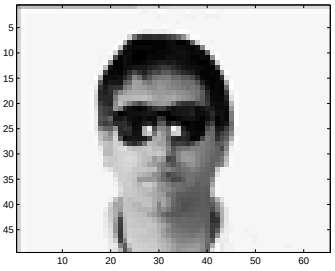
- In this thesis, we have presented the general idea behind PACFs. While we have noted that reducing the number of training sections has dramatically improved computation time of the PACF, much more work may be done to focus on reducing the computational re-



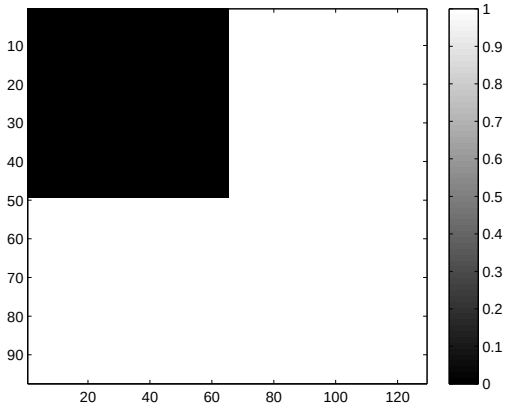
(a) Training Image 1



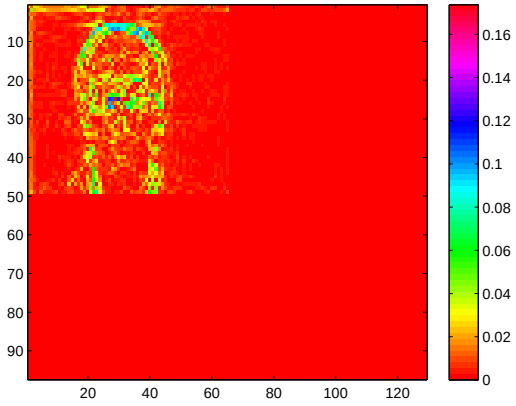
(b) Training Image 2



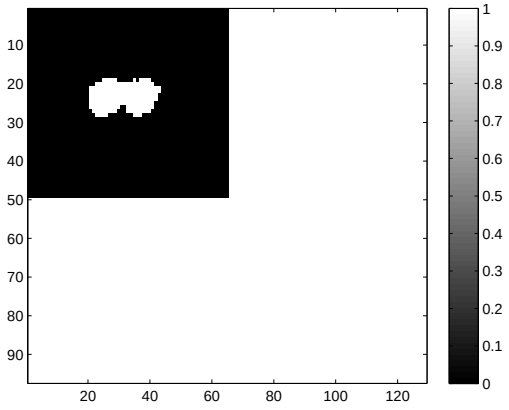
(c) Training Image 3



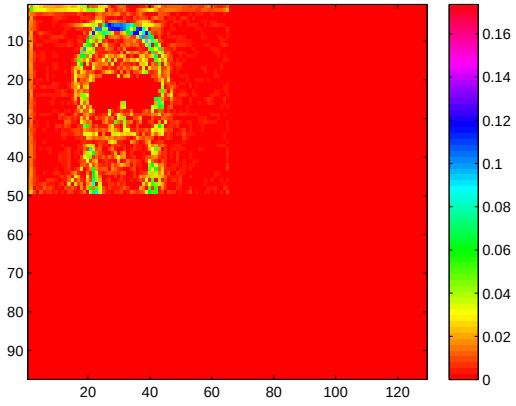
(d) ZA Constraints



(e) Resulting ZACF

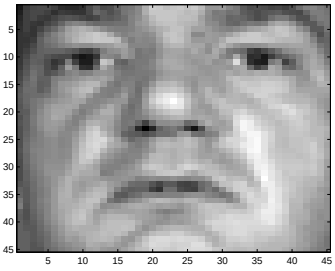


(f) MCF Constraints

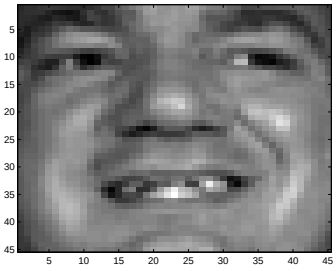


(g) Resulting MCF

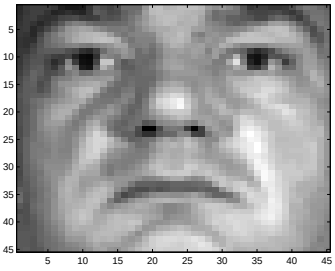
Figure 7.1: An example application of MCF. Three training images are used to build both a ZACF and MCF. Note that the ZACF only constrains the tail, whereas the MCF constrains the tail and the ocular region, which is occluded by sunglasses in the training images. This results in a CF template with a non-standard shape, as it is forced to zero in the occular region. More research is needed to see if recognition performance can be improved by using this method. In these plots, we show the absolute value of the CF templates.



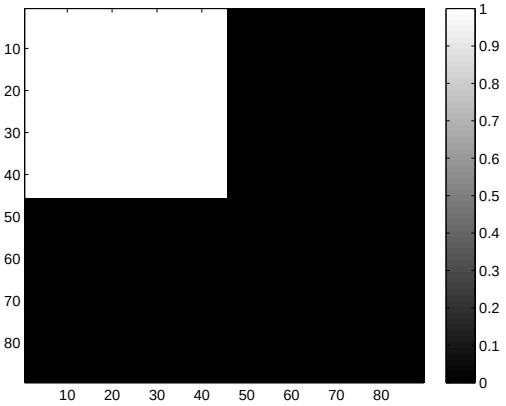
(a) Training Image 1



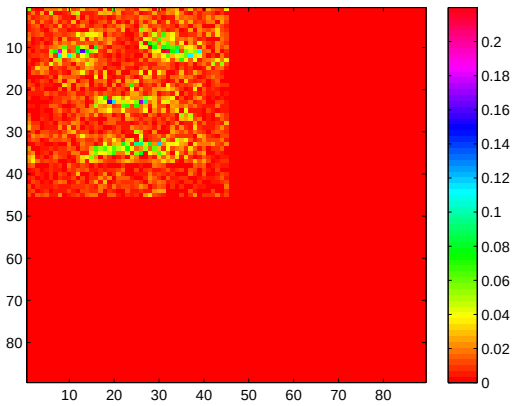
(b) Training Image 2



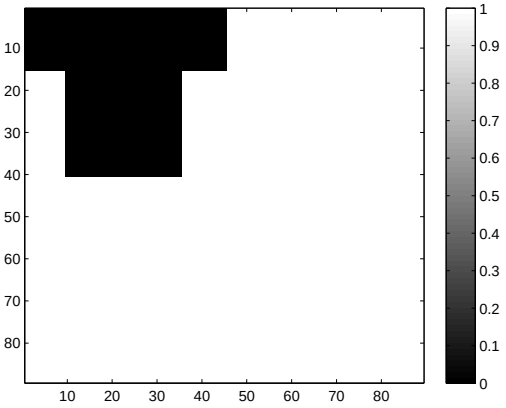
(c) Training Image 3



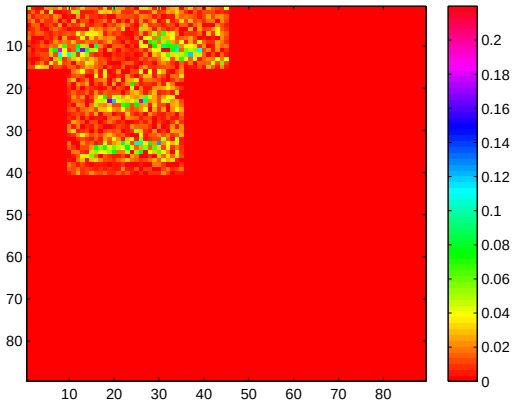
(d) ZA Constraints



(e) Resulting ZACF



(f) MCF Constraints



(g) Resulting MCF

Figure 7.2: A second example application of MCF. Three training images are used to build both a ZACF and MCF. Note that the ZACF only constrains the tail, whereas the MCF constrains the tail and the forces a template that is "T" shaped. More research is needed to see if recognition performance can be improved by using this method. In these plots, we show the absolute value of the CF templates.

quirements for computing PACFs. Unlike ZACFs, the difficulty in training PACF is not only related to the constraints placed on the optimization, but also the large quantity of adjacent section shifts that must be computed to solve for the PACF. Therefore, a method to approximate the PACF may be helpful to reduce training time.

- We have only used a cost function for PACFs that is similar to MOSSE (e.g., minimizing the MSE between the actual correlation output and the desired correlation output). There are other cost functions that may be modified under the general PACF framework. For example, other metrics, such as ONV or ACE may be minimized, and peak equality or inequality constraints may be adopted.
- One goal of future work is to apply PACFs to more realistic data and applications, such as ATR in large images, where a computational/memory improvement can be very important. Right now, one major hurdle to this work is the computational limitations of PACFs, which restrict us to working with small targets and images.
- One direct result of our work on PACFs is another new way of thinking about CF design. In the case of PACF, we demonstrated that it is possible to train CF templates that are smaller than the training signals and achieve good performance. In this case, we did this to reduce the aliasing in each section's output correlation. However, outside of the PACF construct, one can use constraints to build CFs that are smaller than the training images. This, of course, can be combined with the MCF idea, as well. In general, reducing the size of the CF template can offer a significant computational improvement, as the size of the CF template will affect the parameters used for filtering architectures such as OLA and OLS. Although not discussed here, we have also extended this idea to build templates that are larger than the training signals. A larger template could potentially help to reduce sidelobes around correlation peaks and improve recognition performance.

Appendix A

Here, we show that the matrix $\mathbf{R}_q$ is symmetric and Toeplitz. $\mathbf{R}_q$ is an $N_h \times N_h$ matrix, in which the element for row k and column l is given by

$$R_q(k,l) = \sum_{n=-(N_x-1)}^{N_h-1} x_q(k-n)x_q^*(l-n) \tag{A.1}$$

To see why it is symmetric, consider $R_q(a,b)$ and $R_q(b,a)$:

$$R_q(a,b) = \sum_{n=-(N_x-1)}^{N_h-1} x_q(a-n)x_q^*(b-n) \tag{A.2}$$

$$R_q(b,a) = \sum_{n=-(N_x-1)}^{N_h-1} x_q(b-n)x_q^*(a-n) \tag{A.3}$$

By comparing Eq. A.2 and Eq. A.3, we can see that matrix $\mathbf{R}_q$ is conjugate symmetric. For real signals (as is usually the case) $\mathbf{R}_q$ is symmetric. To see why the matrix is Toeplitz, consider the fact that we can describe any diagonal in the matrix by the equation

$$l = k \pm c \tag{A.4}$$

where $c = 0, \ldots, N_h - 1$. Note that c denotes the number of diagonals away from the main diagonal. For example, the main diagonal is described with $c = 0$ ($l = k$), with k varying from 0 to $N_h - 1$. Diagonals below the main diagonal are described by taking the negative sign in Eq.

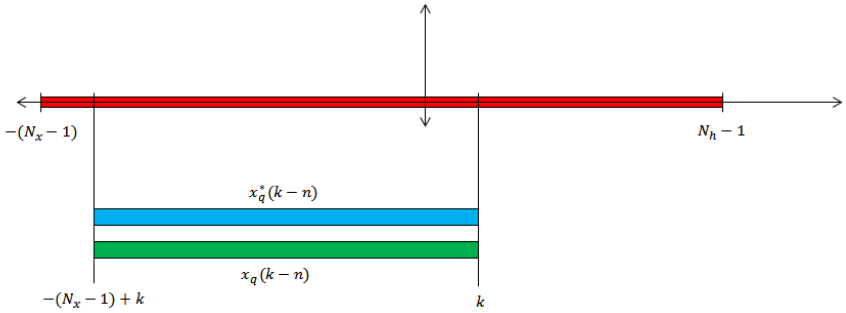
A.4, with k varying from c to $N_h - 1$. Diagonals above the main diagonal are described with the positive sign in Eq. A.4, with k varying from 0 to $N_h - 1 - c$. We can substitute this expression for l into Eq. A.1 and obtain

$$R_q(k, k \pm c) = \sum_{n=-(N_x-1)}^{N_h-1} x_q(k-n)x_q^*(k \pm c - n) \tag{A.5}$$

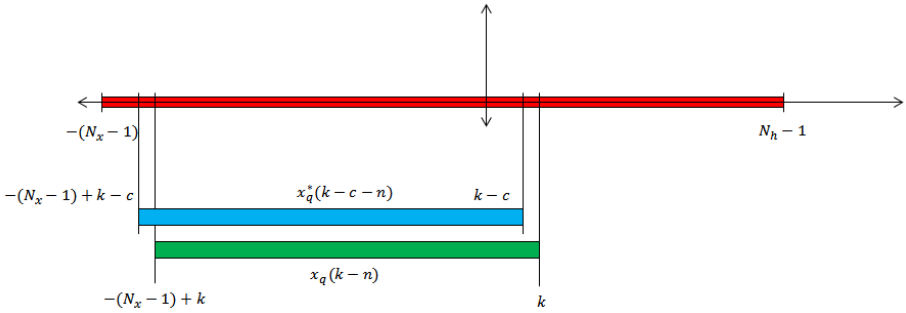
We can think of this as taking two copies of x_q (one conjugated), flipping them, and shifting one by k units to the right and the other (the conjugate) by $k \pm c$ units to the right. The two signals are then multiplied and summed over the time indices $(-(N_x - 1), N_{h-1})$ as found in Eq. A.5. This is illustrated in Figure A.1. Note that the red shaded region represents the limits of the summation. The matrix $\mathbf{R}_q$ is Toeplitz if the result of Eq. A.5 is the same for all values of k for each diagonal. Note that k varies as described above, depending on the diagonal.

- For the main diagonal $(c = 0)$, shown in Figure A.1a, note that the result of Eq. A.5 will always be the same, because the entirety of both signals $x_q^*(k-n)$ and $x_q(k-n)$ always fall in the range over which the sum is computed, regardless of k.
- For off-diagonal elements below the main diagonal (sign of c is negative), shown in Figure A.1b, $x_q^*(k-c-n)$ is shifted by c units to the left relative to $x_q(k-n)$. Although the lower limit of $x_q^*(k-c-n)$ may be outside of the range of the summation, the overlap of the signals, given by the limits $(-(N_x-1)+k, k-c)$, is always within the range of the summation. Therefore, regardless of the value of k, the result of Eq. A.5 will always be the same. Therefore every diagonal below the main diagonal will be equal.
- For the off-diagonal elements above the main diagonal (sign of c is positive), shown in Figure A.1c, $x_q^*(k+c-n)$ is shifted by c units to the right relative to $x_q(k-n)$. Although the upper limit of $x_q^*(k+c-n)$ may be outside of the range of the summation, the overlap of the signals, given by the limits $(-(N_x-1)+k+c, k)$, is always within the range of the summation. Therefore, regardless of the value of k, the result of Eq. A.5 will always be the same. Therefore every diagonal above the main diagonal will be equal.

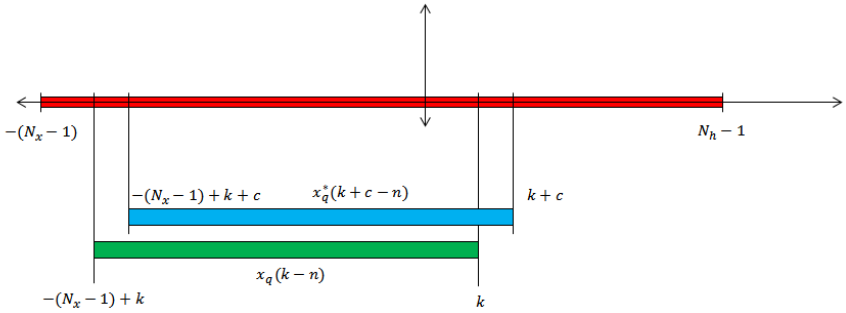
Given the above, the matrix $\mathbf{R}_q$ will always be Toeplitz. This will be useful, because several efficient matrix inversion algorithms exist for Toeplitz matrices.



(a) Main Diagonal



(b) Diagonal Below the Main Diagonal



(c) Diagonal Above the Main Diagonal

Figure A.1: Illustration of summations for matrix $\mathbf{R}_q$.

Appendix B

To gain insight as to why the conventional FDMACE filter yields non-zero values for the entire template, we investigate the FDMACE filter solution and illustrate how it is similar to the ECPSDF. We first present the ECPSDF [8]. The ECPSDF is generally designed in the space domain. First, it seeks a CF template $\mathbf{h}$ such that the inner product of the set of training signals and the template is equal to a constant, i.e.,

$$\mathbf{X}^T\mathbf{h} = \mathbf{u} \tag{B.1}$$

where $\mathbf{X}$ is a $N_x \times Q$ matrix whose columns represent the training signals in the space domain, and $\mathbf{u}$ is a $N_x \times 1$ vector corresponding to the desired peak constraints. Note that we have used 1D notation here for simplicity. Because this constraint alone is an under-determined system, the CF template is assumed to be a linear combination of the training images, i.e.,

$$\mathbf{h} = \mathbf{X}\mathbf{a} \tag{B.2}$$

where the $Q \times 1$ vector $\mathbf{a}$ is a vector of weights. To solve for $\mathbf{a}$, we substitute Eq. B.2 into Eq. B.1, obtaining

$$\mathbf{X}^T\mathbf{X}\mathbf{a} = \mathbf{u}$$

$$\mathbf{a} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{u} \tag{B.3}$$

Substituting Eq. B.3 into Eq. B.2 yields the ECPSDF,

$$\mathbf{h} = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{u} \tag{B.4}$$

The ECPSDF may be given equivalently in the frequency domain as

$$\bar{\mathbf{h}}_{ECPSDF,\bar{\mathbf{X}}} = N\bar{\mathbf{X}}(\bar{\mathbf{X}}^{+}\bar{\mathbf{X}})^{-1}\mathbf{u} \tag{B.5}$$

where N is the DFT size and the columns of $\bar{\mathbf{X}}$ contain the DFTs of the training signals.

Now let us recall the FDMACE formulation as given in Eq. 2.9,

$$\bar{\mathbf{h}} = N\mathbf{D}^{-1}\bar{\mathbf{X}}\left(\bar{\mathbf{X}}^{+}\mathbf{D}^{-1}\bar{\mathbf{X}}\right)^{-1}\mathbf{u} \tag{B.6}$$

Following the same procedure as [65], we can show that the FDMACE may be interpreted as an ECPSDF for when the training signals *and* the test signals are pre-whitened using the transform $\mathbf{D}^{-1/2}$. Because $\mathbf{D}$ is diagonal and positive definite (all elements are real and positive numbers), the FDMACE may be expressed as

$$\bar{\mathbf{h}} = N\mathbf{D}^{-1/2}\mathbf{D}^{-1/2}\bar{\mathbf{X}}\left(\bar{\mathbf{X}}^{+}\mathbf{D}^{-1/2}\mathbf{D}^{-1/2}\bar{\mathbf{X}}\right)^{-1}\mathbf{u}$$

$$\bar{\mathbf{h}} = N\mathbf{D}^{-1/2}\bar{\mathbf{Y}}\left(\bar{\mathbf{Y}}^{+}\bar{\mathbf{Y}}\right)^{-1}\mathbf{u}$$

$$\bar{\mathbf{h}} = \mathbf{D}^{-1/2}\bar{\mathbf{h}}_{ECPSDF,\bar{\mathbf{Y}}} \tag{B.7}$$

where

$$\bar{\mathbf{h}}_{ECPSDF,\bar{\mathbf{Y}}} = N\bar{\mathbf{Y}}\left(\bar{\mathbf{Y}}^{+}\bar{\mathbf{Y}}\right)^{-1}\mathbf{u} \tag{B.8}$$

and

$$\bar{\mathbf{Y}} = \mathbf{D}^{-1/2}\bar{\mathbf{X}} \tag{B.9}$$

Note that $\bar{\mathbf{h}}_{ECPSDF,\bar{\mathbf{Y}}}$ is the ECPSDF trained on the whitened training images, $\bar{\mathbf{Y}}$. The extra $\mathbf{D}^{-1/2}$ term in Eq. B.7 can be interpreted as the whitening operation for the test images. To see why, consider the following. Each value in a test signal's correlation output is given by the inner product of the CF template and the corresponding *test chip* (the region of the test signal corresponding to the location of the correlation value). Because inner products are preserved in the frequency domain, the output correlation value (for test chip $\mathbf{z}$) is given by

$$y_z = \bar{\mathbf{h}}^{+}\bar{\mathbf{z}} \tag{B.10}$$

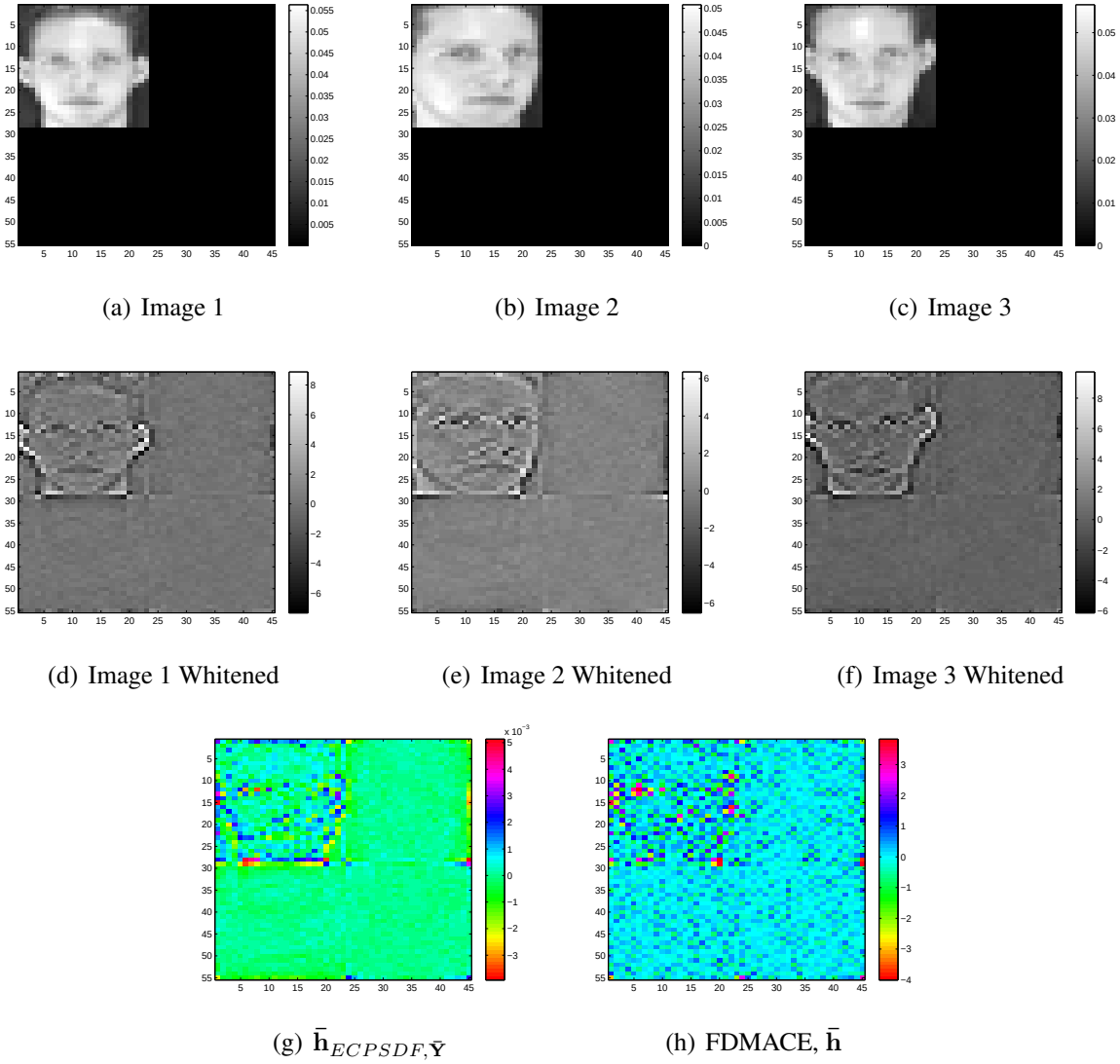
where $\bar{\mathbf{z}}$ is the DFT of the test chip$\mathbf{z}$. For the FDMACE filter, this may be rewritten as

$$\begin{aligned} y_z &= \left(\mathbf{D}^{-1/2}\bar{\mathbf{h}}_{ECPSDF,\bar{\mathbf{Y}}}\right)^{+}\bar{\mathbf{z}} \\ &= \bar{\mathbf{h}}^{+}_{ECPSDF,\bar{\mathbf{Y}}}\mathbf{D}^{-1/2}\bar{\mathbf{z}} \end{aligned} \tag{B.11}$$

where the term $\mathbf{D}^{-1/2}\bar{\mathbf{z}}$ corresponds to the whitened test chip. In practice, of course, each test chip is not explicitly whitened; rather, the whitening matrix $\mathbf{D}^{-1/2}$ is included in the FDMACE filter itself.

With this interpretation of FDMACE, we can see why the FDMACE template is non-zero by examining the whitened training signals $\bar{\mathbf{Y}}$ and $\bar{\mathbf{h}}_{ECPSDF,\bar{\mathbf{Y}}}$, which is a linear combination

of the whitened training images. In Fig. B.1 we show 3 training images of size $N_{x,1} \times N_{x,2} = 28 \times 23$ for an FDMACE filter. We then show the whitened versions of these training images (we transform the columns of $\bar{\mathbf{Y}}$ back to the space domain). Note that the tails of these images no longer contain zeros. Because the FDMACE can be interpreted as a linear combination of these whitened training images, we can see why the tail of the FDMACE template is non-zero. It is for this reason that we must explicitly constrain the tail of the template to zero in our ZAMACE formulation.



(a) Image 1 (b) Image 2 (c) Image 3

(d) Image 1 Whitened (e) Image 2 Whitened (f) Image 3 Whitened

(g) $\bar{\mathbf{h}}_{ECPSDF,\bar{\mathbf{Y}}}$ (h) FDMACE, $\bar{\mathbf{h}}$

Figure B.1: Demonstration of how the FDMACE formulation is essentially an ECPSDF using whitened train and test images. First, we show three training images that are padded prior to training the CF (a-c). Next, we show the whitened versions of these padded images (d-f). Note that the tail of these images are no longer zero. Finally, we show $\bar{\mathbf{h}}_{ECPSDF,\bar{\mathbf{Y}}}$ and the FDMACE, $\bar{\mathbf{h}}$ in g and h, respectively.

Appendix C

Here we show why the matrix $\mathbf{K} = \mathbf{B}^{+}\mathbf{D}^{-1}\mathbf{B}$ is symmetric, positive definite, and real.

Symmetry Here we must show that $\mathbf{K}^{+} = \mathbf{K}$. Note that we may write

$$\begin{aligned} \mathbf{K}^{+} &= \left(\mathbf{B}^{+}\mathbf{D}^{-1}\mathbf{B}\right)^{+} \\ &= \mathbf{B}^{+}(\mathbf{D}^{-1})^{+}\mathbf{B} \\ &= \mathbf{B}^{+}\mathbf{D}^{-1}\mathbf{B} \end{aligned} \tag{C.1}$$

because the matrix $\mathbf{D}$ is real. Therefore, $\mathbf{K}^{+} = \mathbf{K}$.

Positive Definite Here, we show that the matrix K is positive definite. To be positive definite, the condition

$$\mathbf{x}^{+}\mathbf{K}\mathbf{x} > 0 \tag{C.2}$$

must hold for any vector $\mathbf{x}$. This may be rewritten as

$$\mathbf{x}^{+}\mathbf{B}^{+}\mathbf{D}^{-1}\mathbf{B}\mathbf{x} > 0$$

$$\mathbf{y}^{+}\mathbf{D}^{-1}\mathbf{y} > 0 \tag{C.3}$$

where

$$\mathbf{y} = \mathbf{B}\mathbf{x} \tag{C.4}$$

Note that, for any $\mathbf{y} \neq \mathbf{0}$, the expression $\mathbf{y}^{+}\mathbf{D}^{-1}\mathbf{y} > 0$ always is true, because the matrix $\mathbf{D}$ is itself positive definite (it is a diagonal matrix consisting of all real and positive numbers). Therefore, we must only show that $\mathbf{y} = \mathbf{B}\mathbf{x}$ is never equal to the zero vector, for any input vector $\mathbf{x} \neq \mathbf{0}$. In other words, if

$$\mathbf{B}\mathbf{x} = \mathbf{0} \tag{C.5}$$

then $\mathbf{y}^{+}\mathbf{D}^{-1}\mathbf{y} = 0$. However, note that (in one dimension) the matrix $\mathbf{B}$ is of size $N \times p + Q$, where N is the dimension of the DFT, and $p < N$ is the number of zero-aliasing constraints, and Q is the number of training images. Note that in this case, the columns of matrix $\mathbf{B}$ are formed from the conjugates of the rows of the inverse DFT matrix $\mathbf{F}_N^{-1}$, which are orthogonal, as well as the DFTs of Q training signals, which we assume are independent. Therefore, the matrix $\mathbf{B}$ is of rank $p + Q$, and the columns of $\mathbf{B}$ are therefore linearly independent. Therefore, equation C.5 can never be true. As a result, $\mathbf{y}$ is never the zero vector, and matrix $\mathbf{K}$ is positive definite.

Real Here we assume that we are processing real training signals. Recall that $\mathbf{B} = [\ \bar{\mathbf{X}}\quad \mathbf{A}\ ]$. Observe that the matrix may be written as

$$\mathbf{K} = \begin{bmatrix} \bar{\mathbf{X}}^{+} \\ \mathbf{A}^{+} \end{bmatrix} \mathbf{D}^{-1} \begin{bmatrix} \bar{\mathbf{X}} & \mathbf{A} \end{bmatrix}$$
$$= \begin{bmatrix} \bar{\mathbf{X}}^{+}\mathbf{D}^{-1}\bar{\mathbf{X}} & \bar{\mathbf{X}}^{+}\mathbf{D}^{-1}\mathbf{A} \\ \mathbf{A}^{+}\mathbf{D}^{-1}\bar{\mathbf{X}} & \mathbf{A}^{+}\mathbf{D}^{-1}\mathbf{A} \end{bmatrix} \tag{C.6}$$

We aim to show that each of these four submatrices are real. First, note that diagonal matrix $\mathbf{D}$ contains the average power spectrum of the training images. Note that this power spectrum is real and symmetric. Mathematically, this can be written as

$$\mathbf{D}(k,k) = \mathbf{D}(N-k, N-k) \tag{C.7}$$

where k is an index variable. This means that the diagonal of matrix $\mathbf{D}^{-1}$ is also real and symmetric. Next note that we may write

$$\bar{\mathbf{X}} = \mathbf{F}_N \mathbf{X} \tag{C.8}$$

where $\mathbf{X}$ is a $N \times Q$ matrix whose columns contain the N-length padded training signals in the time domain. Note that the columns of $\bar{\mathbf{X}}$ are each conjugate symmetric. Mathematically, if vector $\bar{\mathbf{x}}_i$ represents the ith column of $\bar{\mathbf{X}}$, we can write

$$\bar{\mathbf{x}}_i(k) = \bar{\mathbf{x}}_i^{*}(N-k) \tag{C.9}$$

Let us start with matrix $\bar{\mathbf{X}}^{+}\mathbf{D}^{-1}\bar{\mathbf{X}}$. Note that the ith column of matrix $\mathbf{D}^{-1}\bar{\mathbf{X}}$ contain the element-wise multiplication ($\mathbf{D}^{-1}\bar{\mathbf{x}}_i$) of the diagonal of of $\mathbf{D}^{-1}$ (which is real and symmetric) with the vector $\bar{\mathbf{x}}_i$, which is conjugate symmetric. Therefore, $\mathbf{D}^{-1}\bar{\mathbf{x}}_i$ is a conjugate symmetric vector as well. Next, we can rewrite the matrix as

$$\bar{\mathbf{X}}^{+}\mathbf{D}^{-1}\bar{\mathbf{X}} = \mathbf{X}^{+}\mathbf{F}_{N}^{+}\mathbf{D}^{-1}\bar{\mathbf{X}} \tag{C.10}$$

Note that the N-point IDFT matrix is given by $\frac{1}{N}\mathbf{F}_{N}^{+}$. Therefore, the ith column of matrix $\mathbf{F}_{N}^{+}\mathbf{D}^{-1}\bar{\mathbf{X}}$ contains the (scaled) IDFT of the conjugate symmetric vector $\mathbf{D}^{-1}\bar{\mathbf{x}}_i$. By the properties of the DFT, this is a real signal. Because matrix $\mathbf{X}$ is real, matrix $\bar{\mathbf{X}}^{+}\mathbf{D}^{-1}\bar{\mathbf{X}}$ must be real.

A similar method may be used to show that the other submatrices are real. First, note that $\mathbf{A}^{+}$ is the last $N - N_x$ rows of the $N \times N$ IDFT matrix $\mathbf{F}_{N}^{-1}$. This means $\mathbf{A}$ is the last $N - N_x$ columns of $\mathbf{F}_N$, which we denote $\mathbf{A} = \mathbf{F}_{N-N_x}$. While $\mathbf{F}_{N-N_x}$ is not an actual DFT operation, it shares properties with $\mathbf{F}_N$. Namely, $\mathbf{F}_{N-N_x}^{+}\mathbf{z}$ is always real if $\mathbf{z}$ is a vector that is conjugate symmetric. Therefore, we can show that $\mathbf{A}^{+}\mathbf{D}^{-1}\bar{\mathbf{X}}$ is real by writing it as

$$\mathbf{A}^{+}\mathbf{D}^{-1}\bar{\mathbf{X}} = \mathbf{F}_{N-N_x}^{+}\mathbf{D}^{-1}\bar{\mathbf{X}} \tag{C.11}$$

As before, the columns of $\mathbf{D}^{-1}\bar{\mathbf{X}}$ are conjugate symmetric and therefore $\mathbf{A}^{+}\mathbf{D}^{-1}\bar{\mathbf{X}}$ is real.

For matrices $\bar{\mathbf{X}}^{+}\mathbf{D}^{-1}\mathbf{A} = \mathbf{X}^{+}\mathbf{F}_{N}^{+}\mathbf{D}^{-1}\mathbf{F}_{N-N_x}$ and $\mathbf{A}^{+}\mathbf{D}^{-1}\mathbf{A} = \mathbf{F}_{N-N_x}^{+}\mathbf{D}^{-1}\mathbf{F}_{N-N_x}$, note that the columns of $\mathbf{F}_{N-N_x}$ are conjugate symmetric. This means that the columns of matrix $\mathbf{Z} = \mathbf{D}^{-1}\mathbf{F}_{N-N_x}$ are also conjugate symmetric. The transformations $\mathbf{F}_{N}^{+}\mathbf{Z}$ and $\mathbf{F}_{N-N_x}^{+}\mathbf{Z}$ yield real values. Therefore, both $\bar{\mathbf{X}}^{+}\mathbf{D}^{-1}\mathbf{A}$ and $\mathbf{A}^{+}\mathbf{D}^{-1}\mathbf{A}$ are real. Because all four submatrices are real, $\mathbf{K}$ is also real.

Note that if $\mathbf{D}$ is replaced with $\mathbf{T} = \mathbf{D} + \lambda\mathbf{P}$, all of these proofs still hold, as $\mathbf{P}$ is the power spectral density of the noise model, and is also real and symmetric.

Appendix D

For all three correlation methods (conventional, OLA, and OLS), multidimensional correlation can be optimized by taking advantage of the zero padding required when computing multidimensional DFTs. Rather than zero-padding the signal in all dimensions prior to taking FFTs in each dimension (as was assumed previously), one can dynamically change the ordering of each FFT operation to reduce the number of required multiplications.

To illustrate this idea, we use the conventional case as an example (see Fig. D.1). In this example, we use the same template and signal as was shown in Fig. 5.1. We again show the signal and template in Fig. D.1a and b, respectively. We show the original padding method in Fig. D.1 c and d. With these padded signals, note that it is not necessary to take the DFT of columns (or rows) that are all zero (as the result will be zero). The question, therefore, is whether to take the FFT in the vertical direction first or the horizontal direction first. For example, it would be more optimal to not zero pad in the horizontal direction before taking the FFT of each column (see Figure D.1 e and f) or to not zero pad in the vertical direction before taking the FFT of each row (see Figure D.1 g and h). Because the DFT is linear, either method is acceptable (order does not matter). It is also possible to use a different ordering for the signal than that used for the template. In the 2D case, there are two possible ways to compute the 2D DFT of the template and the signal; therefore, there are 4 total ways to perform the correlation, each of which will result in a different number of multiplies. To avoid this ambiguity, we will use the original padding method (Figure D.1 c and d), zero padding in all dimensions prior to computing

FFTs. This simplification in effect represents the worst-case scenario, a sort of upper limit on the computational complexity. Despite this, we provide an analysis of the alternative padding methods in this section.

To illustrate this idea, we present a generalized notation. Consider a signal x of size $N_{x,1} \times \cdots \times N_{x,D}$. Suppose we want to take an FFT of size $Q_1 \times \cdots \times Q_D$. Next, assume that the order in which we take the FFT is given by $O_x = \{d_1, \ldots, d_D\}$, where $d_p \in \{1, \ldots, D\}$. Note that $d_p \neq d_{p'}$ for $p \neq p'$ (i.e. O_x cannot have repeat entries). For example, if $D = 3$ and $O_x = \{2, 1, 3\}$, the FFT would first be taken along the second dimension, followed by the first dimension and finally the third dimension. Note that the possible sets of O_x are the permutations of the integers from 1 to D, which is given by $D!$.

To quantify the number of computations required under this formulation, we must determine the number of computations needed for the FFTs along each dimension. First, we compute the FFTs along dimension d_1. In this case, we would pad the signal along dimension d_1 such that the new size is Q_{d_1}. The size in the other dimensions are still $N_{x,d} \mid_{d \neq d_1}$. Therefore, the total required computations for the FFTs along dimension d_1 is given by

$$C_{d_1} = \left[\prod_{\gamma=2}^{D} N_{x,d_\gamma} \right] f_M(Q_{d_1}) \tag{D.1}$$

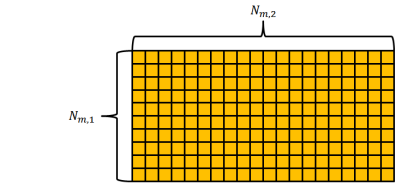
Next, we must pad the signal with zeros along dimension d_2. Prior to the FFT, the signal measures $N_{x,d}$ for dimensions $d \neq d_1$ and $d \neq d_2$ and Q_{d_λ} for dimensions d_λ where $\lambda = 1$ and $\lambda = 2$. Therefore, the total required computations for the FFTs along dimension d_2 is given by

$$C_{d_2} = \left[Q_{d_1} \prod_{\gamma=3}^{D} N_{x,d_\gamma} \right] f_M(Q_{d_2}) \tag{D.2}$$

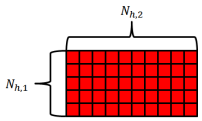
We continue in this manner through all dimensions. For the last dimension (d_D), the signal is padded along the final dimension such that it now measures $Q_1 \times \cdots \times Q_D$. Therefore, the total required computations for the FFTs along dimension d_D is given by

$$C_{d_D} = \left[\prod_{\lambda=1}^{D-1} Q_{d_\lambda} \right] f_M(Q_{d_D}) \tag{D.3}$$

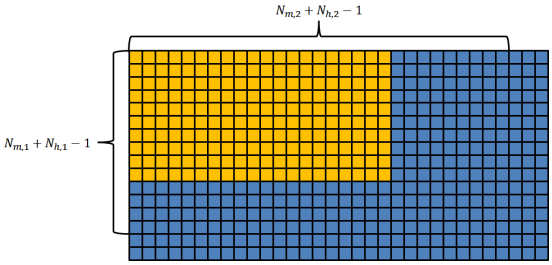
We may generalize these expressions as the number of computations required for the FFT along



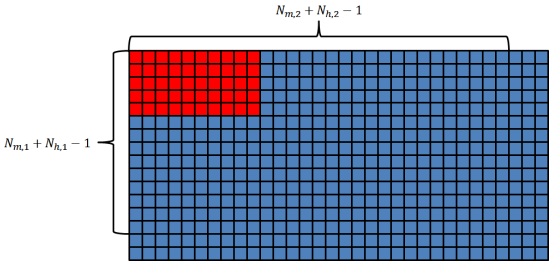
(a) Signal



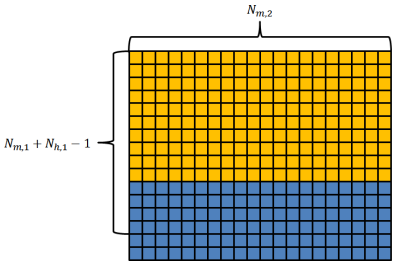
(b) Template



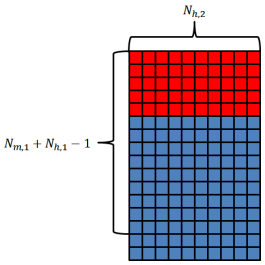
(c) Padded Signal



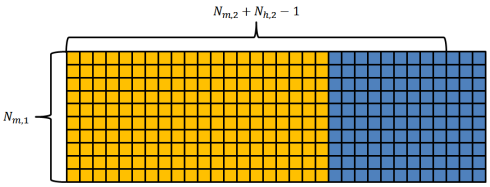
(d) Padded Template



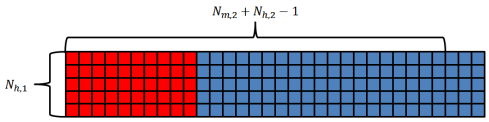
(e) Signal, Vertical Padding



(f) Template, Vertical Padding



(g) Signal, Horizontal Padding



(h) Template, Horizontal Padding

Figure D.1: Demonstration of 2D correlation using different methods for padding the signal and the template.

dimension d_p,

$$C_{d_p} = \left[\prod_{\lambda=1}^{p-1} Q_{d_\lambda}\right]\left[\prod_{\gamma=p+1}^{D} N_{x,d_\gamma}\right] f_M(Q_{d_p}) \tag{D.4}$$

The total complexity of the D-dimensional FFT of signal x is given by

$$C_{FFT}(O_x) = \sum_{p=1}^{D} C_{d_p} \tag{D.5}$$

When computing correlations, note that the IFFT operation (after the element-wise multiplication step) is always the same. At this point in time, both the D dimensional DFTs of both the signal and the template are of the same dimension in the frequency domain ($Q_1 \times \cdots \times Q_D$). Therefore, the complexity of the IFFT operation is given by

$$C_{IFFT}(Q_d) = \sum_{d=1}^{D} \frac{f_M(Q_d)}{Q_d} Q^{\#} \tag{D.6}$$

where

$$Q^{\#} = \prod_{d=1}^{D} Q_d \tag{D.7}$$

Now, we may express the overall complexity of the conventional, OLA, and OLS schemes in terms of these optimized multidimensional FFTs. Note that the optimal FFT order O_h for the template may not be the same as the optimal FFT order O_m for the signal. For the conventional case, we may express the complexity as (compare to equation 5.15)

$$\hat{C}_{conv,D}(N_{c,d}, O_h, O_m) = \min_{O_h} C_{FFT}(O_h) + \min_{O_m} C_{FFT}(O_m) + C_{IFFT}(N_{c,d}) + N_c^{\#} \tag{D.8}$$

where the FFT size Q_d is $N_{c,d}$.

For OLA and OLS, the complexity is given by (compare to equation 5.20)

$$\hat{C}_{OLX,d}(N_d, K_d, O_h, O_s) = \min_{O_h} C_{FFT}(O_h) + \left[\sum_{k_1=0}^{K_1-1} \cdots \sum_{k_D=0}^{K_D-1} \min_{O_s} C_{FFT}(O_s) \right] + K^{\#} C_{IFFT}(N_d) + K^{\#} N^{\#} \tag{D.9}$$

Here, the first term accounts for the DFT of the filter. The second term accounts for the DFT of each section. Note that we express this as D nested sums. The size of each section is in general a function of the section number. Typically, sections are of size $L_1 \times \cdots \times \cdots L_D$, but sections at the end of rows or columns are often smaller (hence our notation). A truly optimized scheme will account for this size difference when computing the DFTs. The third term accounts for the $K^{\#}$ IFFTs (one for each section). Finally, the last term accounts for the element-wise multiplication between the filter and each section.

We have included this discussion for the sake of completeness. However, note that in Chapter 5, we represent the complexity of multidimensional DFTs as was described in Sections 5.3.1 and 5.3.2.

Appendix E

Here, we define matrix $\widetilde{\mathbf{R}}^{N_1 \times N_2}_{\phi_1^{'},\phi_2^{'}}$ for the different cases depicted in Fig. 6.8. Depending on the case (sign of $\phi_1^{'}$ and $\phi_2^{'}$ in Eq. 6.43), $\widetilde{\mathbf{R}}^{N_1 \times N_2}_{\phi_1^{'},\phi_2^{'}}$ will take on a different value. For the sake of notational simplicity, we denote the amount of overlap in each dimension as

$$M_1 = \sigma_1(|\phi_1^{'}|) \tag{E.1}$$

$$M_2 = \sigma_2(|\phi_2^{'}|) \tag{E.2}$$

Recall that we indicate an $a \times b$ matrix of all zeros as $\mathbf{0}_{a \times b}$ and an $a \times a$ identity matrix as $\mathbf{I}_a$.

- **Case 1 and Case 4** ($\phi_1^{'} \leq 0$, $\phi_2^{'} < 0$) **and Case 8** ($\phi_1^{'} > 0$, $\phi_2^{'} > 0$)

  For Case 1 and 4, we shift the values in the lower right corner of the input to the upper left corner of the output. First, we define the submatrices

$$\mathbf{S}_{m0} = \mathbf{0}_{N_1 N_2 \times N_1 N_2 - N_1 M_2} \tag{E.3}$$

$$\mathbf{S}_{m1} = \mathbf{0}_{N_1 N_2 \times N_1 - M_1} \tag{E.4}$$

$$\mathbf{S}_{m2} = \begin{bmatrix} \mathbf{I}_{M_1} \\ \mathbf{0}_{N_1 N_2 - M_1 \times M_1} \end{bmatrix} \tag{E.5}$$

We then define

$$\mathbf{S}_{b0}(u) = \left[\begin{array}{cc} \mathbf{S}_{m1} & CS(\mathbf{S}_{m2}, uN_1, 0) \end{array} \right] \tag{E.6}$$

where the $CS(\mathbf{Z}, u, v)$ operation is a circular shift of matrix $\mathbf{Z}$ u units down and v units to the right. Then, we have

$$\widetilde{\mathbf{R}}^{N_1 \times N_2}_{\phi_1', \phi_2'} = [\begin{array}{cccccc} \mathbf{S}_{m0} & \mathbf{S}_{b0}(0) & \mathbf{S}_{b0}(1) & \cdots & \mathbf{S}_{b0}(M_2 - 1) \end{array}] \tag{E.7}$$

For Case 8 only, we have

$$\widetilde{\mathbf{R}}^{N_1 \times N_2}_{\phi_1', \phi_2'} = [\begin{array}{cccccc} \mathbf{S}_{m0} & \mathbf{S}_{b0}(0) & \mathbf{S}_{b0}(1) & \cdots & \mathbf{S}_{b0}(N_1 M_2 - 1) \end{array}]^T \tag{E.8}$$

because of symmetry.

- **Case 2** ($\phi_1' < 0$, $\phi_2' = 0$)

  Here, we shift the values in the last M_1 rows of the input to the top M_1 rows of the output. We define the submatrices

$$\mathbf{S}_{m3} = \left[\begin{array}{ccc} \mathbf{0}_{M_1 \times N_1 - M_1} & \mathbf{I}_{M_1} & \mathbf{0}_{M_1 \times N_1 N_2 - N_1} \\ \multicolumn{3}{c}{\mathbf{0}_{N_1 - M_1 \times N_1 N_2}} \end{array} \right] \tag{E.9}$$

  We then define

$$\mathbf{S}_{b1}(v) = CS(\mathbf{S}_{m3}, 0, vN_1) \tag{E.10}$$

  Then, we have

$$\widetilde{\mathbf{R}}^{N_1\times N_2}_{\phi_1',\phi_2'} = \begin{bmatrix} \mathbf{S}_{b1}(0) \\ \mathbf{S}_{b1}(1) \\ \vdots \\ \mathbf{S}_{b1}(N_2-1) \end{bmatrix} \tag{E.11}$$

- **Case 3 and Case 5** ($\phi_1' \leq 0$, $\phi_2' > 0$) **and Case 6** ($\phi_1' > 0$, $\phi_2' < 0$)

  For Case 3 and 5, we shift the values in the lower left corner of the input to the upper right corner of the output. First, we define the submatrices

$$\mathbf{S}_{m4} = \mathbf{0}_{N_1M_2\times N_1-M_1} \tag{E.12}$$

$$\mathbf{S}_{m5} = \begin{bmatrix} \mathbf{I}_{M_1} \\ \mathbf{0}_{N_1M_2-M_1\times M_1} \end{bmatrix} \tag{E.13}$$

$$\mathbf{S}_{m6} = \mathbf{0}_{N_1(N_2-M_2)\times N_1N_2} \tag{E.14}$$

$$\mathbf{S}_{m7} = \mathbf{0}_{N_1M_2\times N_1(N_2-M_2)} \tag{E.15}$$

  We then define

$$\mathbf{S}_{b2}(u) = \begin{bmatrix} \mathbf{S}_{m4} & CS(\mathbf{S}_{m5}, uN_1, 0) \end{bmatrix} \tag{E.16}$$

  Then, we have

$$\widetilde{\mathbf{R}}^{N_1\times N_2}_{\phi_1',\phi_2'} = \begin{bmatrix} \mathbf{S}_{m6} \\ \mathbf{S}_{b2}(0) & \mathbf{S}_{b2}(1) & \cdots & \mathbf{S}_{b2}(M_2-1) & \mathbf{S}_{m7} \end{bmatrix} \tag{E.17}$$

For Case 6 only, we have

$$\widetilde{\mathbf{R}}_{\phi_1^{'},\phi_2^{'}}^{N_1\times N_2} = \begin{bmatrix} & \mathbf{S}_{m6} & & \\ \mathbf{S}_{b2}(0) & \mathbf{S}_{b2}(1) & \cdots & \mathbf{S}_{b2}(M_2-1) \quad \mathbf{S}_{m7} \end{bmatrix}^T \tag{E.18}$$

because of symmetry.

- **Case 7** ($\phi_1^{'} > 0$, $\phi_2^{'} = 0$)

  Here, we shift the values in the first M_1 rows of the input to the last M_1 rows of the output. We define the submatrices

$$\mathbf{S}_{m8} = \begin{bmatrix} \mathbf{0}_{N_1-M_1\times N_1N_2} \\ \mathbf{I}_{M_1} \quad \mathbf{0}_{M_1\times N_1N_2-M_1} \end{bmatrix} \tag{E.19}$$

  We then define

$$\mathbf{S}_{b3}(v) = CS(\mathbf{S}_{m8}, 0, vN_1) \tag{E.20}$$

  Then, we have

$$\widetilde{\mathbf{R}}_{\phi_1^{'},\phi_2^{'}}^{N_1\times N_2} = \begin{bmatrix} \mathbf{S}_{b3}(0) \\ \mathbf{S}_{b3}(1) \\ \vdots \\ \mathbf{S}_{b3}(N_2-1) \end{bmatrix} \tag{E.21}$$

Examples Here, we illustrate how the selection matrix $\widetilde{\mathbf{R}}_{\phi_1^{'},\phi_2^{'}}^{N_1\times N_2}$ works for each of the cases shown in Fig. 6.8. Note that $\widetilde{\mathbf{R}}_{\phi_1^{'},\phi_2^{'}}^{N_1\times N_2}$ operates on a vectorized array. As an example, we will use the 5×4 array

$$\mathbf{W} = \begin{bmatrix} 1 & 6 & 11 & 16 \\ 2 & 7 & 12 & 17 \\ 3 & 8 & 13 & 18 \\ 4 & 9 & 14 & 19 \\ 5 & 10 & 15 & 20 \end{bmatrix} \tag{E.22}$$

This is vectorized column-by-column as

$$\mathbf{w} = vect(\mathbf{W}) = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & \cdots & 10 & \cdots & 20 \end{bmatrix}^T \tag{E.23}$$

Where we denote the vectorization operation as $vect()$. We also define the unvectorization operation as $unvect()$, i.e. $\mathbf{W} = unvect(\mathbf{w})$.

- **Case 1**

  In this case, we shift the values in matrix $\mathbf{W}$ from the lower right hand corner to the upper left hand corner of the output matrix. For example, if the amount of overlap is $(3, 2)$, this results in

$$unvect(\widetilde{\mathbf{R}}^{N_1 \times N_2}_{\phi_1', \phi_2'} \mathbf{w}) = \begin{bmatrix} 13 & 18 & 0 & 0 \\ 14 & 19 & 0 & 0 \\ 15 & 20 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{E.24}$$

- **Case 2**

  In this case, we shift the values in the last rows of matrix $\mathbf{W}$ to the first rows of the output matrix. For example, if the amount of vertical overlap is 3, this results in

$$unvect(\widetilde{\mathbf{R}}^{N_1 \times N_2}_{\phi_1', \phi_2'} \mathbf{w}) = \begin{bmatrix} 3 & 8 & 13 & 18 \\ 4 & 9 & 14 & 19 \\ 5 & 10 & 15 & 20 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{E.25}$$

- **Case 3**

  In this case, we shift the values in matrix $\mathbf{W}$ from the lower left hand corner to the upper right hand corner of the output matrix. For example, if the amount of overlap is $(3, 2)$, this results in

$$unvect(\widetilde{\mathbf{R}}^{N_1 \times N_2}_{\phi_1', \phi_2'} \mathbf{w}) = \begin{bmatrix} 0 & 0 & 3 & 8 \\ 0 & 0 & 4 & 9 \\ 0 & 0 & 5 & 10 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{E.26}$$

- **Case 4**

  In this case, we shift the values in the right most columns of matrix $\mathbf{W}$ to the left most columns of the output matrix. For example, if the amount of overlap in the horizontal direction is 2, this results in

$$unvect(\widetilde{\mathbf{R}}^{N_1 \times N_2}_{\phi_1', \phi_2'} \mathbf{w}) = \begin{bmatrix} 11 & 16 & 0 & 0 \\ 12 & 17 & 0 & 0 \\ 13 & 18 & 0 & 0 \\ 14 & 19 & 0 & 0 \\ 15 & 20 & 0 & 0 \end{bmatrix} \tag{E.27}$$

- **Case 5**

  In this case, we shift the values in the left most columns of matrix $\mathbf{W}$ to the right most columns of the output matrix. For example, if the amount of overlap in the horizontal direction is 2, this results in

$$unvect(\widetilde{\mathbf{R}}^{N_1 \times N_2}_{\phi_1', \phi_2'} \mathbf{w}) = \begin{bmatrix} 0 & 0 & 1 & 6 \\ 0 & 0 & 2 & 7 \\ 0 & 0 & 3 & 8 \\ 0 & 0 & 4 & 9 \\ 0 & 0 & 5 & 10 \end{bmatrix} \tag{E.28}$$

- **Case 6**

  In this case, we shift the values in a matrix $\mathbf{W}$ from the upper right hand corner to the lower left hand corner of the output matrix. For example, if the amount of overlap is $(3, 2)$, this is accomplished as

$$unvect(\widetilde{\mathbf{R}}^{N_1 \times N_2}_{\phi_1', \phi_2'} \mathbf{w}) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 11 & 16 & 0 & 0 \\ 12 & 17 & 0 & 0 \\ 13 & 18 & 0 & 0 \end{bmatrix} \tag{E.29}$$

- **Case 7**

  In this case, we shift the values in the first rows of matrix $\mathbf{W}$ to the last rows of the output matrix. For example, if the amount of vertical overlap is 3, this results in

$$unvect(\widetilde{\mathbf{R}}^{N_1 \times N_2}_{\phi_1', \phi_2'} \mathbf{w}) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 6 & 11 & 16 \\ 2 & 7 & 12 & 17 \\ 3 & 8 & 13 & 18 \end{bmatrix} \tag{E.30}$$

- **Case 8**

  In this case, we shift the values in a matrix $\mathbf{W}$ from the upper left hand corner to the lower right hand corner of the output matrix. For example, if the amount of overlap is $(3, 2)$, this is accomplished as

$$unvect(\widetilde{\mathbf{R}}^{N_1 \times N_2}_{\phi_1', \phi_2'} \mathbf{w}) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 6 \\ 0 & 0 & 2 & 7 \\ 0 & 0 & 3 & 8 \end{bmatrix} \tag{E.31}$$

Bibliography

[1] B. V. K. Vijaya Kumar, A. Mahalanobis, and R. D. Juday, *Correlation Pattern Recognition*. Cambridge, UK: Cambridge University Press, 2005. 1, 2.3, 3.8, 5, 5.3.3

[2] M. D. Rodriguez, J. Ahmed, and M. Shah, "Action MACH a spatio-temporal maximum average correlation height filter for action recognition," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2008, pp. 1–8. 1.1, 2.1.4, 5.1

[3] S. Ali and S. Lucey, "Are correlation filters useful for human action recognition?" in *International Conference on Pattern Recognition (ICPR)*, 2010, pp. 2608–2611. 1.1, 2.1.4, 5.1

[4] A. Mahalanobis, R. Stanfill, and K. Chen, "A bayesian approach to activity detection in video using multi-frame correlation filters," *Proceedings of SPIE*, vol. 8049, 2011. 1.1, 2.1.4

[5] A. Haq, I. Gondal, and M. Murshed, "Action recognition using spatio-temporal distance classifier correlation filter," in *International Conference on Digital Image Computing Techniques and Applications*, 2011, pp. 474–479. 1.1, 2.1.4, 5.1

[6] J. A. Fernandez and B. V. K. Vijaya Kumar, "Space-time correlation filters for human action detection," in *IS&T/SPIE Electronic Imaging, Video Surveillance and Transportation Imaging Applications*, vol. 8663, 2013. 1.1, 2.1.4, 5.1

[7] A. Rodriguez and B. V. K. Vijaya Kumar, "Generalized linear correlation filters," in *SPIE Conference on Automatic Target Recognition*, vol. 8744, 2013. 2.1, 3, 3.4, 3.4.2, 3.4.4

[8] C. F. Hester and D. Casasent, "Multivariant technique for multiclass pattern recognition," *Applied Optics*, vol. 19, no. 11, pp. 1758–1761, 1980. 2.1.1, B

[9] A. Mahalanobis, B. V. K. Vijaya Kumar, and D. Casasent, "Minimum average correlation energy filters," *Applied Optics*, vol. 26, no. 17, pp. 3633–3640, 1987. 2.1.1, 2.1.5, 2.4, 2.4.1, 3

[10] G. Ravichandran and D. Casasent, "Minimum noise and correlation energy optical correlation filter," *Applied Optics*, vol. 31, no. 11, pp. 1823–1833, 1992. 2.1.1, 3

[11] D. Casasent, G. Ravichandran, and S. Bollapragada, "Gaussian–minimum average correlation energy filters," *Applied Optics*, vol. 30, no. 35, pp. 5176–5181, 1991. 2.1.1, 3

[12] B. V. K. Vijaya Kumar, A. Mahalanobis, S. Song, S. R. F. Sims, and J. F. Epperson, "Minimum squared error synthetic discriminant functions," *Optical Engineering*, vol. 31, no. 5, pp. 915–922, 1992. 2.1.1, 3

[13] B. V. K. Vijaya Kumar, "Minimum-variance synthetic discriminant functions," *J. Opt. Soc. Am. A*, vol. 3, no. 10, pp. 1579–1584, 1986. 2.1.1, 3.3

[14] P. Refregier, "Filter design for optical pattern recognition: multicriteria optimization approach," *Optics Letters*, vol. 15, no. 15, pp. 854–856, 1990. 2.1.1, 3, 3.3, 3.3, 3.6.4

[15] Refregier and J. Figue, "Optimal trade-off filter for pattern recognition and their comparison with weiner approach," *Optical Computing and Processing*, no. 1(3), pp. 245–266, 1991. 2.1.1, 3, 3.3

[16] A. Mahalanobis, B. V. K. Vijaya Kumar, S. Song, S. R. F. Sims, and J. F. Epperson, "Unconstrained correlation filters," *Applied Optics*, vol. 33, no. 17, pp. 3751–3759, 1994. 2.1.2, 3, 3.4.1, 3.5, 3.6.4, 3.8.3

[17] B. V. K. Vijaya Kumar, A. Mahalanobis, and D. W. Carlson, "Optimal trade-off synthetic discriminant function filters for arbitrary devices," *Optics Letters*, vol. 19, pp. 1556–1558, 1994. 2.1.2, 3, 3.6.4

[18] M. Alkanhal, A. Mahalanobis, and B. V. K. Vijaya Kumar, "Improving the false alarm capabilities of the maximum average correlation height correlation filter," *Optical Engineering*, vol. 39, no. 5, pp. 1133–1141, 2000. 2.1.2, 3

[19] B. V. K. Vijaya Kumar and M. Alkanhal, "Eigen-extended maximum average correlation height (eemach) filters for automatic target recognition," *Proceedings of SPIE*, vol. 4379, pp. 424–431, 2001. 2.1.2, 3

[20] D. S. Bolme, B. A. Draper, and J. R. Beveridge, "Average of synthetic exact filters," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 2105–2112. 2.1.2, 2.1.4, 2.1.5, 2.4, 3, 3.6, 3.6.1, 3.6.1, 4.5.3

[21] D. S. Bolme, J. R. Beveridge, B. A. Draper, and Y. M. Lui, "Visual object tracking using adaptive correlation filters," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2010, pp. 2544–2550. 2.1.2, 2.1.4, 3, 3.6, 3.6.3, 3.6.3, 3.6.4, 6.2.1

[22] W. H. Zhou Lubing, "Facial landmark localization via boosted and adaptive filters," *International Conference on Image Processing*, pp. 519–523, 2013. 2.1.2, 3, 3.6.1

[23] A. Rodriguez, V. N. Boddeti, B. V. K. Vijaya Kumar, and A. Mahalanobis, "Maximum margin correlation filter: A new approach for localization and classification," *IEEE Transactions on Image Processing*, vol. 22, no. 2, pp. 631–643, 2013. 2.1.3, 2.1.4, 3, 3.7

[24] B. V. K. Vijaya Kumar, M. Savvides, and C. Xie, "Correlation pattern recognition for face recognition," *Proceedings of the IEEE*, vol. 94, no. 11, pp. 1963–1976, 2006. 2.1.4

[25] M. Alkanhal, G. Muhammad, A. Alotaibi, and K. Alqahtani, "A robust recognition system for partially occluded faces," in *Proceedings of the Conference on Image and Vision Computing New Zealand*, 2012, pp. 475–479. 2.1.4

[26] J. Thornton, M. Savvides, and B. V. K. Vijaya Kumar, "A Bayesian approach to deformed pattern matching of iris images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 4, pp. 596–606, 2007. 2.1.4

[27] V. N. Boddeti, F. Su, and B. V. K. Vijaya Kumar, "A biometric key-binding and template protection framework using correlation filters," *International Conference on Biometrics*,

vol. 5558, pp. 919–929, 2009. 2.1.4

[28] V. N. Boddeti and B. V. K. Vijaya Kumar, "A framework for binding and retrieving class-specific information to and from image patterns using correlation filters," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 9, pp. 2064–2077, 2013. 2.1.4, 7.3

[29] K. Miyazawa, K. Ito, T. Aoki, K. Kobayashi, and H. Nakajima, "An effective approach for iris recognition using phase-based image matching," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 10, pp. 1741–1756, 2008. 2.1.4

[30] H. Nakajima, K. Kobayashi, and T. Higuchi, "A fingerprint matching algorithm using phase-only correlation," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 87, no. 3, pp. 682–691, 2004. 2.1.4

[31] K. Ito and T. Aoki, "Phase-based image matching and its application to biometric recognition," *Proceedings of APSIPA Annual Summit and Conference*, pp. 1–7, 2013. 2.1.4

[32] A. Rodriguez and B. V. K. Vijaya Kumar, "Automatic target recognition of multiple targets from two classes with varying velocities using correlation filters," in *17th IEEE International Conference on Image Processing*, 2010, pp. 2781–2784. 2.1.4

[33] R. Kerekes and B. V. K. Vijaya Kumar, "Enhanced video-based target detection using multi-frame correlation filtering," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 45, no. 1, pp. 289–307, 2009. 2.1.4

[34] D. S. Bolme, Y. M. Lui, B. A. Draper, and J. R. Beveridge, "Simple real-time human detection using a single correlation filter," in *International Workshop on Performance Evaluation of Tracking and Surveillance*, 2009, pp. 1–8. 2.1.4

[35] V. N. Boddeti, T. Kanade, and B. V. K. Vijaya Kumar, "Correlation filters for object alignment," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 2291–2298. 2.1.4, 7.3

[36] Y. Li, Z. Wang, and H. Zeng, "Correlation filter: An accurate approach to detect and locate low contrast character strings in complex table environment," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 12, pp. 1639–1644, 2004. 2.1.4

[37] T. S. H. Kiani Galoogahi and S. Lucey, "Multi-channel correlation filters," *IEEE International Conference on Computer Vision*, pp. 4321–4328, 2013. 2.1.4

[38] S. I. Sudharsanan, A. Mahalanobis, and M. K. Sundareshan, "Unified framework for the synthesis of synthetic discriminant functions with reduced noise variance and sharp correlation structure," *Optical Engineering*, vol. 29, no. 9, pp. 1021–1028, 1990. 2.1.5, 2.4, 2.4.2

[39] A. Rodriguez and B. V. K. Vijaya Kumar, "Dealing with circular correlation effects," in *SPIE Conference on Automatic Target Recognition*, vol. 8744, 2013. 2.1.5, 2.4

[40] H. Bristow, A. Eriksson, and S. Lucey, "Fast convolutional sparse coding," *2013 IEEE Conference on Computer Vision and Pattern Recognition*, vol. 0, pp. 391–398, 2013. 2.1.5, 7.3

[41] J. A. Fernandez and B. V. K. Vijaya Kumar, "Zero-aliasing correlation filters," *International*

Symposium on Image and Signal Processing and Analysis, pp. 101–106, 2013. 2.1.5, 3.8.1, 6.2.2, 6.3.1

[42] J. G. Proakis and D. G. Manolakis, *Digital Signal Processing*. Upper Saddle River, NJ: Pearson Prentice Hall, 2007. 2.2, 5, 5.3, 5.3.3, 6.1

[43] F. S. Samaria and A. C. Harter, "Parameterisation of a stochastic model for human face identification," in *IEEE Workshop on Applications of Computer Vision*, 1994, pp. 138–142. 2.3, 3.2, 3.8, 4.1.3, 4.2.4, 4.4, 4.5.1

[44] G. Moody and R. Mark, "The impact of the MIT-BIH arrhythmia database," *IEEE Engineering in Medicine and Biology Magazine*, vol. 20, no. 3, pp. 45–50, 2001. 2.4.3

[45] G. H. Golub, "Some Modified Matrix Eigenvalue Problems," *SIAM Review*, vol. 15, no. 2, pp. 318–334. 3.5.2

[46] M. Savvides, K. Venkataramani, and B. V. K. Vijaya Kumar, "Incremental updating of advanced correlation filters for biometric authentication systems," in *International Conference on Multimedia and Expo*, vol. 3, 2003, pp. 229–232. 3.6.4

[47] A. Rodriguez, *Maximum Margin Correlation Filters*. PhD Thesis, Carnegie Mellon University, 2012. 3.7, 3.7.2, 3.7.2

[48] A. Beck and M. Teboulle, "A fast iterative shrinkage-thresholding algorithm for linear inverse problems," *SIAM Journal on Imaging Sciences*, vol. 2, no. 1, pp. 183–202, 2009. 4.3, 4.3, 4.4

[49] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, UK: Cambridge University Press, 2004. 4.3

[50] P. Phillips, P. Flynn, T. Scruggs, K. Bowyer, J. Chang, K. Hoffman, J. Marques, J. Min, and W. Worek, "Overview of the face recognition grand challenge," in *IEEE Conference on Computer Vision and Pattern Recognition*, vol. 1, 2005, pp. 947–954. 4.5.1, 4.5.1.2, 7.3

[51] Military Sensing Information Analysis Center, "ATR algorithm development image database," www.sensiac.org, 2011. [Online]. Available: www.sensiac.org 4.5.2

[52] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2005, pp. 886–893. 4.5.2

[53] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000. 4.5.3

[54] P. Phillips, H. Moon, S. Rizvi, and P. Rauss, "The FERET evaluation methodology for face-recognition algorithms," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 10, pp. 1090–1104, 2000. 4.5.3

[55] C. Dubout and F. Fleuret, "Exact acceleration of linear object detectors," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2012, Oral, pp. 301–311. 5.1

[56] M. Narasimha, "Modified overlap-add and overlap-save convolution algorithms for real signals," *Signal Processing Letters, IEEE*, vol. 13, no. 11, pp. 669–671, 2006. 5.1

[57] H.-C. Chiang and J.-C. Liu, "Fast algorithm for FIR filtering in the transform domain," *Signal Processing, IEEE Transactions on*, vol. 44, no. 1, pp. 126–129, 1996. 5.1

[58] G. Burel, "A matrix-oriented approach for analysis and optimisation of block digital filters,"

Proceedings of WSEAS International Conference on Signal, Speech, and Image Processing, 2002. 5.1

[59] A. Daher, E. H. Baghious, G. Burel, and E. Radoi, "Overlap-save and overlap-add filters: Optimal design and comparison," *IEEE Transactions on Signal Processing*, vol. 58, no. 6, pp. 3066–3075, 2010. 5.1, 6.2

[60] H. Sasaki, Z. Li, and H. Kiya, "FFT-based full-search block matching using overlap-add method," in *Picture Coding Symposium (PCS)*, 2010, pp. 586–589. 5.1, 1, 5.5, 5.5, 2, 5.5

[61] J. A. Fernandez and B. V. K. Vijaya Kumar, "Multidimensional overlap-add and overlap-save for correlation and convolution," *International Conference on Image Processing*, pp. 509–513, 2013. 5.1, 5.6

[62] W.-H. Chang and T. Nguyen, "On the fixed-point accuracy analysis of FFT algorithms," *Signal Processing, IEEE Transactions on*, vol. 56, no. 10, pp. 4673–4682, 2008. 5.3

[63] M. Savvides, B. V. K. Vijaya Kumar, and P. K. Khosla, "Eigenphases vs eigenfaces," in *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, vol. 3, 2004, pp. 810–813 Vol.3. 6.2

[64] A. Martinez and R. Benavente, "The ar face database," *Computer Vision Center*, no. 24, June 1998. 7.3

[65] B. V. K. Vijaya Kumar and A. Mahalanobis, "Alternate interpretation for minimum variance synthetic discriminant functions," *Applied Optics*, vol. 25, no. 15, pp. 2484–2485, August 1986. B