Fast Line Segment Extraction for Line Features and 3D Reconstruction

Dissertation

der Mathematisch-Naturwissenschaftlichen Fakultät
der Eberhard Karls Universität Tübingen
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
(Dr. rer. nat.)

vorgelegt von
Benjamin Wassermann
aus Filderstadt

Tübingen
2025

Gedruckt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät
der Eberhard Karls Universität Tübingen.

Tag der mündlichen Qualifikation:	16.03.2026
Dekan:	Prof. Dr. Thilo Stehle
1. Berichterstatter:	Prof. Dr. Andreas Schilling
2. Berichterstatter:	Prof. Dr. Gottfried Zimmermann

"$E = mc^2$"

Albert Einstein

Abstract

Edge detection and line segment extraction are fundamental tools in computer vision, enabling applications such as object recognition, scene segmentation, and 3D reconstruction. This thesis investigates methods for structuring raw edge data into meaningful clusters for line segment extraction. The focus lies on the detection of meaningful edge regions, the extraction of connected edge points, the processing of line segments, as well as optimization and validation methods to improve the accuracy and robustness of line segments.

After introducing important fundamentals of edge processing, various edge response techniques are described, analyzed, and compared based on their ability to preserve detail while minimizing noise. In addition, operations such as non-maximum suppression and zero-crossing are examined for their role in improving edge localization and the reliability of edge segment extraction. To reduce noise and fragmentation in edge segment detection and improve accuracy, robust methods for edge region growing and edge point connection are introduced, including validation methods.

Beyond edge detection, structured line segments provide a higher level of geometric data representation. This is particularly useful for 3D reconstruction and indoor navigation, where structural regularity is strong but texture information is limited. Line segment detection techniques, including line fitting methods, are evaluated for their role in representing precise geometric structure and supporting higher-level visual analysis, such as line feature extraction and matching. Computational efficiency will also be evaluated, with a comparative analysis of runtime performance and optimization strategies to balance accuracy and processing speed.

This thesis also provides an introduction to the open source framework that has evolved from the implementations of the various methods described in this thesis. In addition to the methods, it provides an analyzer application to directly apply and visualize the methods, as well as tools to perform evaluations on the methods. It lays an important milestone for future research in edge and line segment-based image analysis tasks.

Zusammenfassung

Die Erkennung von Kanten und die Extraktion von Liniensegmenten sind grundlegende Werkzeuge der Computer Vision, die Anwendungen wie Objekterkennung, Szenensegmentierung und 3D-Rekonstruktion ermöglichen. In dieser Arbeit werden Methoden zur Strukturierung von Kantenrohdaten in sinnvolle Cluster für die Extraktion von Liniensegmenten untersucht. Der Schwerpunkt liegt auf der Erkennung aussagekräftiger Kantenregionen, der Extraktion zusammenhängender Kantenpunkte, der Verarbeitung von Liniensegmenten sowie auf Optimierungs- und Validierungsmethoden zur Verbesserung der Genauigkeit und Robustheit von Liniensegmenten.

Nach einer Einführung in die Grundlagen der Kantenverarbeitung werden verschiedene Edge-Response-Techniken beschrieben und hinsichtlich ihrer Fähigkeit, Details zu erhalten und gleichzeitig das Rauschen zu minimieren, analysiert und verglichen. Zusätzlich werden Operationen wie Non-Maxima-Supression und Zero-Crossing auf ihre Rolle bei der Verbesserung der Kantenlokalisierung und der Zuverlässigkeit der Kantensegmentextraktion untersucht. Um das Rauschen und die Fragmentierung bei der Erkennung von Kantensegmenten zu reduzieren und die Genauigkeit zu verbessern, werden robuste Methoden für das Wachsen von Kantenregionen und die Verbindung von Kantenpunkten eingeführt, einschließlich Validierungsmethoden.

Neben der Kantenerkennung bieten strukturierte Liniensegmente eine höhere Ebene der geometrischen Datendarstellung. Dies ist besonders nützlich für die 3D-Rekonstruktion und Navigation in Innenräumen, wo die strukturelle Regelmäßigkeit hoch ist, die Texturinformationen jedoch begrenzt sind. Techniken zur Erkennung von Liniensegmenten, einschließlich Methoden zur Linienausrichtung, werden hinsichtlich ihrer Rolle bei der Darstellung präziser geometrischer Strukturen und bei der Unterstützung visueller Analysen auf höherer Ebene, wie z. B. der Extraktion von Linienmerkmalen und der daraus resultierenden Fähigkeit, gleiche Segmente wiederzufinden, bewertet. Die Effizienz der Berechnung wird ebenfalls bewertet, mit einer vergleichenden Analyse der Laufzeitleistung und Optimierungsstrategien, um ein Gleichgewicht zwischen Genauigkeit und Verarbeitungsgeschwindigkeit zu erreichen.

Diese Arbeit bietet außerdem eine Einführung in das Open-Source-Framework, das aus den Implementierungen der verschiedenen in dieser Arbeit beschriebenen Methoden entstanden ist. Zusätzlich zu den Methoden stellt es eine Analyseanwendung zur Verfügung, mit der die Methoden direkt angewendet und visualisiert werden können, sowie Werkzeuge zur Durchführung von Evaluierungen der Methoden. Damit stellt es einen wichtigen Meilenstein für zukünftige Forschungsarbeiten im Bereich der kanten- und liniensegmentbasierten Bildanalyse dar.

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my supervisor, Prof. Andreas Schilling, for his invaluable guidance and support throughout my journey in the interesting field of computer vision. His trust in my ability to work independently gave me the freedom to explore new ideas, while his insightful discussions and scientific intuition were instrumental in shaping the concepts presented in this thesis. Without his constant encouragement and expertise, completing this thesis in its current form would not have been possible. I am also particularly grateful for his unwavering support and constant reminders to complete this project.

I would also like to express my deepest gratitude to Prof. Gottfried Zimmermann, who played an important role in the early stages of my Ph.D. journey. His generous support, constant availability, and willingness to provide guidance whenever needed helped me overcome the initial challenges of my research. His encouragement and invaluable advice laid the foundation for my academic progress, and I am truly grateful for the time and effort he invested in mentoring me.

Furthermore, I like to extend my appreciation to all past and present members of the Visual Computing and Computer Graphics Group in Tübingen for their support and collaboration. In particular, the fruitful discussions with Manuel Lange have greatly influenced my work. I am also thankful to Roland Wahl and Marlon Mylo from 3D Reality Maps for their valuable discussions, the data they provided, and their technical assistance. My sincere thanks go to Fabian Gorschlüter and Fabian Schweinfurth for their invaluable help in further developing the line extraction tooling and for assisting me with various other tasks during my time as a Ph.D. student.

Additionally, I'am grateful to Daniel Wassermann and Hans-Jörg Frasch for their careful proofreading of parts of this thesis. I also acknowledge the financial support provided by the DFG, which made my Ph.D. studies possible.

Lastly, I want to express my deepest gratitude to my family for their unwavering emotional support. I am especially thankful to my parents, Peter and Cornelie, my brothers and sisters, my parents-in-law, and my children. Most importantly, I extend my heartfelt thanks to my wife, Christina, whose patience, encouragement, and understanding have been a constant source of strength throughout this journey.

Contents

Abstract **v**

Acknowledgements **vii**

Contents **ix**

1 Introduction **1**
1.1 Edges in Human and Computer Vision 1
1.2 What are Edges? 1
1.3 Human Vision 2
1.3.1 Perception of the Human Eye 2
1.3.2 Optical Illusions 4
1.3.3 Edge Detection and the Visual Cortex 5
1.4 Edge Detection in Computer Vision 6
1.4.1 Edges in Images 6
1.4.2 Basic Edge Extraction 7
1.4.3 What Comes Next 8
1.5 Overview 8

2 Mathematical Principles for Image Processing **9**
2.1 Linear Filters and Convolution 9
2.1.1 Separable Filters 10
2.2 Image Smoothing 10
2.2.1 Gaussian Filter 10
2.3 The Derivatives 13
2.3.1 The Image Gradient 13
2.3.2 The Image Laplacian 13
2.3.3 Approximate Image Derivatives with Linear Filters 14
2.3.4 Combine Smoothing and Derivatives 14
2.3.5 Effect on Edges 15
2.4 Scale Space 15
2.4.1 Differences of Gaussians (DOG) 15
2.4.2 Wavelets 17

3 Image Preprocessing **19**
3.1 Information of Color 19
3.2 Upscaling 20
3.3 Image Denoising 20
3.3.1 Gaussian Smoothing 20
3.3.2 Bilateral Filters 21
3.3.3 Non-local Mean 21
3.3.4 Other Methods 21
3.4 Results and Discussion 21

3.4.1 Significance of Color Information . . . 21
3.4.2 Denoising . . . 24

4 Edge Region Detectors **29**
4.1 Derivative Filters . . . 29
4.1.1 Canny Filter . . . 31
4.1.2 Canny-Based Filter Extensions . . . 31
4.1.3 Marr & Hildreth Filter . . . 34
4.1.4 Normalization and DC Response in Derivative Filters . . . 34
4.1.5 Local Line Fitting . . . 35
4.1.6 Conclusion . . . 35
4.2 Local Energy and Phase Congruency . . . 35
4.2.1 Quadrature Filter . . . 35
4.2.2 Combining Multiple Directions . . . 38
4.2.3 Steerable Filters . . . 39
4.2.4 Phase Congruency (PC) . . . 42
4.2.5 Monogenic Signal . . . 44
4.2.6 Conclusion . . . 46
4.3 Statistical Methods . . . 47
4.3.1 SUSAN Operator . . . 47
4.3.2 Conclusion . . . 48
4.4 Morphological Methods . . . 48
4.4.1 RCMG Operator . . . 49
4.4.2 Conclusion . . . 50
4.5 Even more Methods . . . 50
4.5.1 Global Methods . . . 50
4.5.2 Corner Regions . . . 51
4.6 Results and Discussion . . . 51

5 Edge Region Detector Comparison **53**
5.1 Runtime Comparison . . . 53
5.1.1 Hardware Setup . . . 55
5.1.2 Analysis . . . 55
5.1.3 GPU Acceleration . . . 58
5.2 Visual Results . . . 62
5.3 Sensitivity to Noise . . . 63
5.4 Gradient Orientation Precision . . . 69
5.5 Discussion . . . 79

6 Edge Region Operations **81**
6.1 Threshold Methods . . . 81
6.1.1 Global Threshold . . . 81
6.1.2 Double Thresholds . . . 81
6.1.3 Local Thresholds . . . 83
6.1.4 Threshold Estimation . . . 84
6.2 Edge Region Thinning . . . 85
6.2.1 Morphological Thinning . . . 85
6.2.2 Non Maximum Suppression . . . 86
6.2.3 Zero Crossing . . . 89
6.3 Sub-Pixel Precision . . . 91
6.3.1 Subpixel Accuracy for NMS (Devernay Approach) . . . 91

6.3.2 Subpixel Accuracy for ZC . . . 92
6.4 Results and Discussion . . . 94
6.4.1 Threshold Methods . . . 94
6.4.2 NMS and ZC Results . . . 96

7 Connected Edge Components **107**
7.1 Edge Drawing . . . 107
7.2 Edge Linking . . . 109
7.2.1 Basic Algorithm . . . 109
7.2.2 Graph Extension . . . 112
7.3 Edge Pattern Linking . . . 112
7.4 False Edge Segment Detection . . . 113
7.4.1 Contrast based Segment Validation . . . 114
7.4.2 Orientation based Segment Validation . . . 116
7.4.3 Further Refinements . . . 117
7.5 Results and Discussion . . . 117
7.5.1 Edge Segment Detection (ESD) . . . 118
7.5.2 Meaningful Edge Segments . . . 121

8 Line Segment Extraction **123**
8.1 Line Fitting Methods . . . 123
8.1.1 RANSAC . . . 123
8.1.2 Hough Transform . . . 124
8.1.3 Clustered Random Hough . . . 127
8.2 Edge Region Methods . . . 128
8.2.1 The Burns Method . . . 128
8.2.2 Gioi Line Segment Detector (LSD) . . . 130
8.3 Linked Edge Methods . . . 133
8.3.1 Least Square Fit and Extend . . . 133
8.3.2 Split and Merge . . . 134
8.3.3 Primitive Patterns . . . 136
8.4 Line Segment Fitting Methods . . . 138
8.4.1 Linear Regression . . . 138
8.4.2 Principal Component Analysis . . . 139
8.4.3 M-Estimators . . . 140
8.5 False Line Segment Detection . . . 141
8.6 Line Segment Optimization . . . 141
8.7 Results and Discussion . . . 142
8.7.1 Edge Segment Line Fitting Methods . . . 143
8.7.2 Edge Segment Split Methods . . . 144
8.7.3 Line Segment Detectors . . . 145

9 Line Features **149**
9.1 Point vs. Line Features . . . 149
9.2 Line Descriptors . . . 151
9.2.1 Descriptor Composition . . . 152
9.2.2 Left Right Descriptor . . . 152
9.2.3 Orthogonal Descriptor . . . 153
9.2.4 Linked Descriptor . . . 153
9.2.5 Descriptor Matching . . . 154
9.3 Line Feature Matching . . . 154

9.3.1 Matching Candidate Filtering by Constraints 155
9.3.2 Final Line Descriptor Matches 156
9.3.3 Outlier Detection 156
9.4 Results and Discussion 157

10 Line Extraction Framework **159**
10.1 Line Extraction C++ Libraries 159
10.1.1 Framework Dependencies 160
10.1.2 Project Structure 160
10.1.3 Framework Structure 161
10.2 Framework Design 161
10.2.1 Filter Architecture 162
10.2.2 Edge Tools 164
10.2.3 Line Segment Detector Architecture 167
10.3 Analyzer Application 169
10.3.1 Overview 169
10.3.2 Line Profile Analyzer 172
10.3.3 Precision Optimizer 175
10.3.4 Quiver Tool 179
10.3.5 Customization and Extension 180
10.4 Evaluation Tooling 181
10.4.1 Task Architecture 182
10.4.2 Task Runner Architecture 185
10.4.3 Eval Application 186
10.5 Python Tools 187
10.5.1 Python Binding Modules 187
10.5.2 Binding Technique 188
10.5.3 Jupyter Notebooks 190
10.6 Further Information 191

11 Outlook **193**
11.1 Line Features 193
11.2 3D Reconstruction 194
11.3 AI Approaches 194

A Appendix **197**
A.1 DC Component 197
A.2 Phase 200

List of Figures **203**

List of Tables **211**

List of Abbreviations **215**

Bibliography **217**

For my wife Christina...

Chapter 1

Introduction

This thesis presents a comprehensive analysis of line and edge segment extraction methods, emphasizing the underlying mathematical principles and algorithms used in computer vision. The primary goal is to provide a thorough understanding of the techniques used to extract line and edge segments from images, along with their applications in various domains. To achieve this goal, this chapter first introduces the reader to the basic concepts so that the reader can then delve deeper into the subject matter in the following chapters. An overview is provided at the end of this chapter.

1.1 Edges in Human and Computer Vision

Depending on the scientific discipline, computer vision can be approached from two sides. From a biological point of view, the goal of computer vision is to model the visual perception of insects, animals, or humans. From the engineering point of view, the goal is to design autonomous systems that can mimic and eventually outperform human vision in many tasks. To achieve this goal, however, a deep biological understanding is important, because 'vision' is not only the perception and recording of a series of images, but also the extraction of spatial and temporal information. While the first discipline can be handled quite efficiently with today's computer systems, biological systems still easily outperform computer vision for the second discipline. Looking at recent approaches, however, biological understanding combined with the availability of highly parallel computing units has led to promising new approaches (see section 11.3 for more details about new approaches).

The extraction of spatial or temporal information constitutes a higher-level process that typically builds upon fundamental image features. These features, which include boundaries or edges in images, provide valuable information about the composition of the scene captured in the camera snapshot.

1.2 What are Edges?

Edges are perceived as discontinuities in images caused by sharp changes in color or brightness. In the environment, edges are caused by different properties of objects, such as surface orientation or reflectance discontinuity. This happens, for example, when the surface color, texture, or structure of a material changes. Illumination changes can also cause discontinuities such as shadows or highly reflective spots. The most important edges are depth discontinuities caused by occluding objects. They contain most of the geometric information, while the other edges can easily lead to misinterpretation of the environment (such as reflections or shadows). The depth discontinuities have some additional properties:

- As long as the camera can hold the focus, the edge strength or sharpness remains stable.
- The foreground side of the edge is more stable when the camera's perspective changes.

These properties can help to distinguish the occluding edges from other edges, improving the results of e.g. stereo vision methods.

Edges are needed to extract information about the projected environmental geometry for use in higher-level processing such as object recognition or environmental orientation. The projection is done by the human eye or by a camera. A simple pinhole camera collects light from a scene through a small hole and creates an inverted image on the wall opposite the hole. Depending on the exposure time, the intensity of the image can be controlled. Modern cameras can also control the light by adjusting the opening (aperture) and using lenses to focus the light onto the sensor. By moving the lens, they can also change the focus [86, chapter 5.7.6]. The result of the projected environment is an image defined by the intensity function $I(x,y)$, which returns the intensity value (or values, for color information) for any given point on the 2D plane. For digital images, the intensity function is partitioned into finite regions $I(xi,yi)$, and the intensity for each region is determined within a quantization range by the average discrete value, typically $[0,255]$.

In human vision, contours arise not only from continuous edges or boundaries, but also from grouping or continuous features such as points or lines [109]. While the human visual system seems to be trimmed to find patterns and continuous shapes, these tasks are more challenging for computer vision and feature extraction.

1.3 Human Vision

Human vision allows individuals to perceive color and brightness intensities. From this information, the human brain easily detects discontinuities that are used to recognize known objects. Since this is an essential ability to interact and navigate in an environment, the human brain is extremely efficient at extracting this information.

1.3.1 Perception of the Human Eye

Like a camera, the human eye projects 3D space onto a 2D plane. The lens of the eye can focus objects at different distances, and the pupil regulates how much light can pass through the eye, like the aperture of a camera. While the camera must move the lens to focus objects, the human eye changes the curvature of the lens [72]. The resulting image is a 2D representation of the 3D scene. For monochrome images, it can be represented as a continuous intensity function $I(x,y)$, as defined in the previous section.

In this representation, edges can be found in discontinuities of the intensity values. In computer vision, some low-level image preprocessing is done (e.g., smoothing to reduce noise) before edge detection methods are applied (see section 2.2 for more information). Human vision uses massive parallel processing and multiple edge detection mechanisms located in the primary visual cortex of the brain. But before the visual signals are received by the visual cortex, some low-level image processing

(such as smoothing and edge enhancement) is also performed on the retina of the eye [16, 87, 66].

The retina of an eye can be divided into 3 successive layers:

1. Rods and cones: Rods and cones are the photoreceptors of the retina. They receive light, become excited, and send a signal to the bipolar cells and the horizontal cell network. Rods are more sensitive to brightness, while cones are more sensitive to color.

2. Bipolar cells: Collect signals from photoreceptors, weight them, and send them to ganglion and amacrine cells. They also receive signals from the horizontal cell network.

3. Ganglion cells: Collect signals from bipolar and amacrine cells. From there, the information is transmitted through the optic nerves to the primary visual cortex.

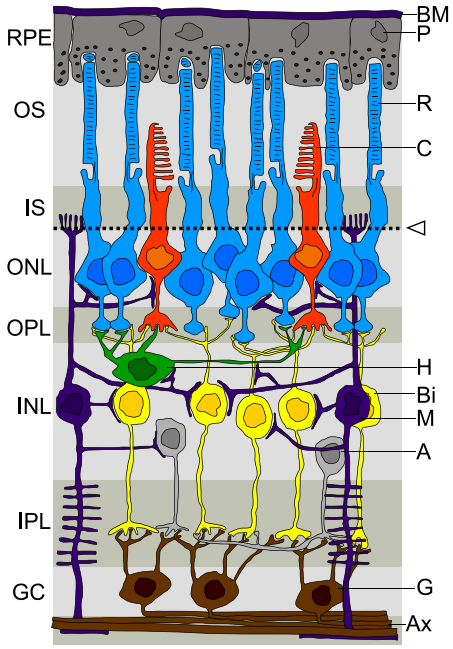


Figure 1.1: Cell types in different layers of a human retina. R: rods; C: cones; H: horizontal cells; Bi: bipolar cells; A: amacrine cells; G: ganglion cells.

This is only a rough outline of the retina. For the cell types mentioned, there are various subspecies that lead to more layers that can be distinguished (see Figure 1.1)[1] and a very complex interaction of signal processing and different types of signals transmitted to different areas of the brain. However, from neurophysiological experiments it is known that the bipolar cells receive information from several photoreceptors connected by horizontal cells, similarly as the ganglion cells receive information from the bipolar cells and a network of amacrine cells. The area of cells

[1] By Peter Hartmann at de.wikipedia, edited by Marc Gabriel Schmid Creating SVG version by Юкатан - Eigenes Werk, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=27237061

or photoreceptors contributing to the information is limited to a small part of the retina. This contributing area is assumed to be circular or elliptical and represents the receptive field of a cell.

It is also known that signals from photoreceptors located in the central region of a bipolar cell's receptive field contribute positively, while those in the outer region contribute inversely. Another distinction is made between on-center and off-center bipolar cells. They differ in the stimulus they receive. The stimulus for the on-central cells is maximal for a bright spot on a dark background, as opposed to the maximal stimulus for the off-central cells, which is a dark spot on a bright background.

A ganglion cell collects this antagonistic information from different bipolar cells. It is assumed that the contribution decreases smoothly with distance from the receiving cell and is interpreted as a Gaussian distribution. This results in a positive Gaussian distribution with a smaller variance from the central inputs and a negative Gaussian distribution with a larger variance from the outer inputs. This suggests that the resulting input to a ganglion cell is derived from a difference of Gaussians (DOG).

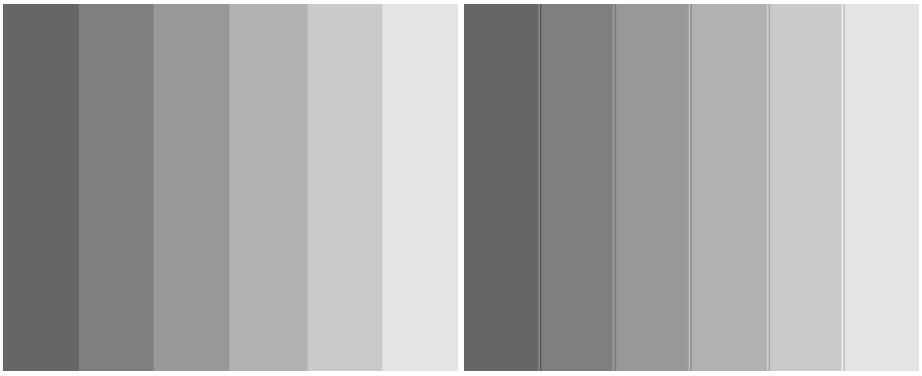


Figure 1.2: At the edges of grayscale bars with different intensities from dark to light, the human eye interprets a non-existent gradient. By applying the Laplacian of Gaussian operator to the Mach band image, the edges are enhanced, making the image sharper and separating each gray bar from the others by overshooting and undershooting at each step of intensity change.

The DOG is used to represent the classical receptive field model. However, this model is only a rough approximation of the preprocessing actually performed by the retina. Recent experimental observations indicate that many photoreceptors outside the classical receptive field may also contribute to ganglion cell sensation. This retinal preprocessing step in human vision enhances edges and allows us to detect thin and weak edges. In computer vision, the DOG is a band-pass filter and can be used to find edges in images (see section subsection 2.4.1 for more information). The preprocessing of the retina also leads to some optical illusions in human vision, such as the Mach band illusion [178] or the Cornsweet illusion [176].

1.3.2 Optical Illusions

Already in 1865, without all the knowledge we have today, Ernst Mach was able to empirically visualize the receptive field model in the retina. He analyzed the optical illusion known today as the Mach band (see Figure 1.2). From these findings, he predicted a Laplacian operation for brightness stimulation of a retinal point, similar

to the DOG. This may be true, since the DOG can be approximated by a Laplacian of Gaussian (LOG) [147, 142] for a certain ratio of variances of the Gaussian. By subtracting the computed DOG or LOG of the image from the original image, an edge enhanced image can be processed when applied by computer vision.

Mach also worked with other optical illusions, but he could not explain all of them with the classical receptive field model. For example, the response of the DOG to the white effect illusion or the checkerboard illusion gives a brightness perception opposite to human visual perception. Ghosh et al. [65] introduced a non-classical receptive field model that extends the classical receptive field by a disinhibitory surround beyond the classical field. Although they assume nonlinear subunits in the receptive field, [170] modeled the response of the extended receptive field as a linear combination of three zero-mean Gaussians at different scales. With this approach they can explain the white effect illusion and the checkerboard illusion.

1.3.3 Edge Detection and the Visual Cortex

Biological physiology provides an understanding of how and where information is transmitted from the retinal ganglion cells and how this information is processed in the human brain. With approximately 80% of the connected ganglion cells, the geniculostriate pathway is the primary route for retinal information. The first stop on this pathway in the brain is the lateral geniculate nucleus (LGN). The LGN has six layers that separate information by eye and cell type. It organizes visual information for the striate cortex (primary visual cortex V1), which is the second stop on the pathway. The striate cortex can send information back to the LGN, so the LGN can be thought of as a kind of regulator of visual information.

Within the striate cortex, cells are assumed to be tuned to different orientations. A bar of light causes different stimulus responses from the cells, depending on the intensity and orientation. The responses are then transmitted to other areas of the brain. In addition, there are different types of cells that vary in complexity. As a result, human vision is quite sensitive to different orientations, while the cells of animals, e.g. a cat, often distinguish between much fewer orientations. The mapping of spatial information from the ganglion cells of the retina is preserved. Thus, information about edge orientation and position can be passed from striate cortex to other regions of the brain that perform motion detection or object recognition [110, chapter 8].

Based on this knowledge, edge detection approaches have been developed using neural networks to model cortical areas of the brain [200, 136, 189, 78]. Since this is a highly parallel process, it can't be easily mapped to today's computer architectures, and processing on typical computer systems remains quite time consuming. Two decades ago, classical computer vision approaches to edge detection were also cumbersome, but they can now be easily computed in real time, even for high-resolution images. In the future, as more highly parallel computing systems become available, more sophisticated approaches may be a better choice for computer vision. Real-time processing is a fundamental requirement for most applications in robotics and computer vision, especially when they use embedded systems and don't have access to cloud systems. Therefore, classical edge detection approaches are typically used and optimized for specific problems.

1.4 Edge Detection in Computer Vision

A fundamental task in computer vision is to extract features to perform higher-level analysis on images, similar to human vision. While significant points within an image, e.g. derived from corners [82] or stable areas (invariance to scale, rotation, illumination, etc.) [134], are preferred because of the simplicity of the geometric type, contours and edges are much more expressive in describing real objects in the image. By approximating contours with linear segments, simple geometric types can be constructed that more accurately represent the scene. With increasing computing power, more complex approaches are emerging to extract features in real-time applications using combined features such as points and line segments [173]. Additional information to build the descriptors for features (for efficient identification within multiple images) is extended by using the phase or other local characteristics [32] instead of simply relying on the image gradient.

1.4.1 Edges in Images

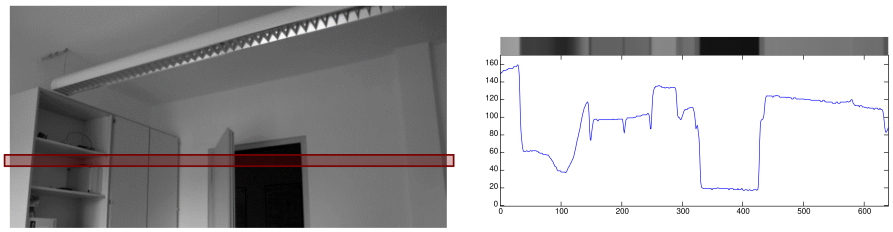


Figure 1.3: On the right side, the 1D intensity function is plotted for the selected scan line of the image on the left side.

To find contours and edges in images, they must be characterized in order to locate them efficiently. As mentioned in the first section of this chapter, edges are discontinuities in the intensity function $I(x,y)$ that represents the image. By choosing a one-dimensional function $f(x)$ that can be extracted from a single row of the image plane, the profiles of edges can be visualized (see Figure 1.3). By analyzing the profiles, two basic edge types can be distinguished:

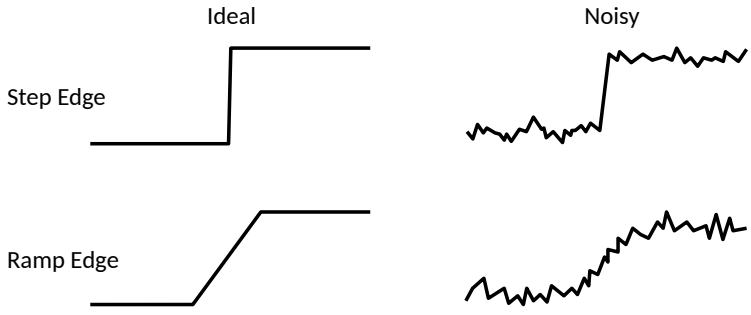


Figure 1.4: Examples of step and ramp edges as ideal and noisy signals.

- Step edge (see Figure 1.4): If the intensity changes by a certain amount, a step edge is observed. With further differentiation, it is also possible to distinguish between step edges and ramp edges. In real images, most step edges will look like a smoothed ramp edge.

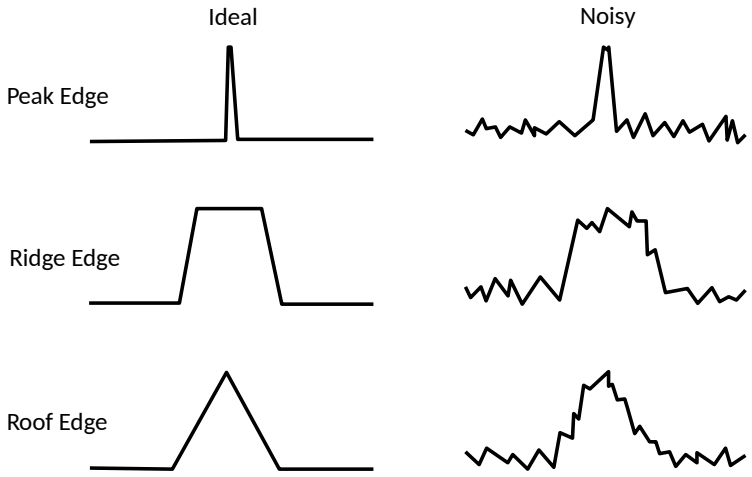


FIGURE 1.5: Examples of ideal and noisy peak, ridge, and roof edge signals.

- Ridge edges (see Figure 1.5): Two opposite intensity changes within a short distance are interpreted as a ridge edge. If the distance is very short, it can also be called a peak edge, or if the intensity changes merge into each other, it is often identified as a roof edge.

The step edge represents a classic edge, while the ridge edge represents a line. Since a line has a width, a scale can be determined. To detect lines in multiple images at different scales, either filters of different sizes must be used, or a filter must be applied to different image sizes. Scale spaces are discussed in more detail in the next chapter.

1.4.2 Basic Edge Extraction

Locating edges in images requires several steps. The steps can vary depending on the method used, but generally there are three main processes:

1. Since most methods are sensitive to noise (and non-synthetic images always contain noise), the first step is often to smooth or denoise the image, e.g. by applying a low-pass filter. Image smoothing and alternatives are discussed in section 2.2.

2. In the second step, the edge regions are extracted by applying linear or nonlinear operators that respond to intensity changes in the image. These methods can be derivative filters, morphological operators, or region-analysis heuristics. The result is an energy response that indicates the probability that a point in the image corresponds to an edge. Depending on the method used, additional information such as orientation or phase is also provided. A detailed overview of existing methods can be found in chapter 4 and chapter 5.

3. Optionally, the final locations of the edges are determined by their maximum response. Since not all maxima found correspond to real edges (e.g. maxima with very low response), a thresholding scheme is applied to remove weak maxima. See chapter 6 for more information.

1.4.3 What Comes Next

After extracting the edge regions, geometric primitives such as points at corners and junctions as well as connected edge points or regions can be processed (see chapter 7 and chapter 8). These connected edge components serve as the basis for line segment extraction, which further simplifies the representation of object boundaries.

Based on these processed features, objects can be tracked across a series of images, or correspondences within multiple images can be identified (e.g., in a stereo setup to estimate the depth map). This can help to solve structure from motion (SfM) or 3D reconstruction problems for robotics and computer vision tasks.

1.5 Overview

After this introduction, the mathematical principles of image processing and edge detection are presented. It covers topics such as linear filters, image smoothing methods, image derivatives, and the concept of scale space (chapter 2). The following chapter deals with image preprocessing issues such as noise reduction or grayscale conversion (chapter 3). Then a detailed overview of existing edge detection methods is given. These include classical methods such as Sobel or Laplace, but also more modern approaches such as Phase Congruency (chapter 4). This is followed by a comparison of the methods presented (chapter 5).

The next chapter deals with the question of what to do with the edge information. It deals with methods that cleverly discard unimportant information in order to obtain the actual edge points (chapter 6). Based on the edge points, the next chapter presents methods for grouping points into segments in a meaningful and efficient way (chapter 7). This is followed by methods to upgrade the edge segments to line segments, as well as some independent approaches to extract line segments from edge detector results (chapter 8).

The next chapter describes how line segments are used to extract line features from images and how they can be used to find similar line features extracted from other images (chapter 9). This is followed by an introduction to the line extraction framework developed in this thesis, which also provides an analysis application and evaluation tools (chapter 10). Finally, there is an outlook on existing follow ups of this work, possible further topics, and recent developments in this area (chapter 11).

At this point I would like to mention that the tool DeepL Write[2] was used to correct spelling, grammar and sentence structure errors in this work.

[2] https://www.deepl.com/write

Chapter 2

Mathematical Principles for Image Processing

This chapter introduces the mathematical principles of image processing and edge detection. This includes linear filters, image smoothing methods, image derivatives, and the concept of scale space.

Image smoothing and image derivatives are commonly used in the classic edge detection process. Smoothing can help reduce the detection of false edges, while derivative filters generate the necessary edge responses for further processing of the final edges.

The scale-space concept, on the other hand, can be used to extract edges from images, but also allows a more general analysis of edges in images, e.g., for scale-invariant image feature descriptors (see chapter 9).

2.1 Linear Filters and Convolution

Since classical image smoothing and image derivation methods rely on linear filters, a brief introduction to this topic is given [205, chapter 3.2]. A linear filter is implemented as a convolution. A convolution consists of a function f and the impulse response of the filter, defined as a function h. The response c of the convolution of f and h is defined as

$$c = f * h \tag{2.1}$$

The convolution has the following properties:

- Commutative: $a * b = b * a$
- Associative: $(a * b) * c = a * (b * c)$.

This will be very useful, because it will be possible to rearrange computations in the most convenient way and improve efficiency. The convolution operation can be thought of as a shift operation, so that each value of c is the sum of the products of h with the shifted version of f:

$$c(x) = f(x) * h(u) = \int_{-\infty}^{\infty} f(x-u)\,h(u)\,du \tag{2.2}$$

In discrete form, for two finite arrays f and h with $N = size(h)$ and $size(f) > N$, the convolution can be written as

$$c(x) = \sum_{u=-N/2}^{N/2} f(x-u)h(u) \tag{2.3}$$

For the two-dimensional case, such as the intensity function $I(x,y)$, the convolution is defined as

$$C(x,y) = I(x,y) * h(u,v) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x-u, y-v)h(u,v)dudv \tag{2.4}$$

In the discrete case:

$$C(x,y) = \sum_{u=-N/2}^{N/2}\sum_{v=-N/2}^{N/2} f(x-u, y-v)h(u,v) \tag{2.5}$$

2.1.1 Separable Filters

A two-dimensional filter with impulse response $h(x,y)$ is separable along the X- and Y-axes, if the following condition is satisfied:

$$h(u,v) = h_x(u)h_y(v) \tag{2.6}$$

The filtering applied to an image is defined as:

$$C(x,y) = I(x,y) * h(u,v) = (I(x,y) * h_x(u)) * h_y(v) \tag{2.7}$$

This property reduces the complexity of convolving a two-dimensional filter of size N from N^2 to $2N$ and also allows recursive implementation.

2.2 Image Smoothing

The calculation of image derivatives can be done with linear filters (see next section). Since this is a local approximation, the computed derivatives are susceptible to noise in the signal (see section 3.3 closer look at noise in images). To overcome this problem, the image is smoothed before the derivative operators are applied. The smoothing process is a low-pass filter applied to the image that removes high frequencies that are also noise. This can also be done with a linear filter. Usually a combined filter is applied that includes both smoothing and derivative.

2.2.1 Gaussian Filter

A simple filter for smoothing would be a mean filter by calculating the average of the neighboring values. This can be realized as a linear filter by applying an *nxn* filter mask to an image, filled with the value $1/n$ at each position, and is usually called a box filter. The abrupt truncation of the mask has unfortunate effects in the frequency domain. They cause high frequency noise, which is called the Gibbs phenomenon [88] (see Figure 2.2 in the middle). To avoid this problem, a weighting can be applied, i.e. reducing the values of the mask further away from the center. This results in small changes at the mask boundaries and prevents the Gibbs phenomenon as seen in Figure 2.2 on the right.

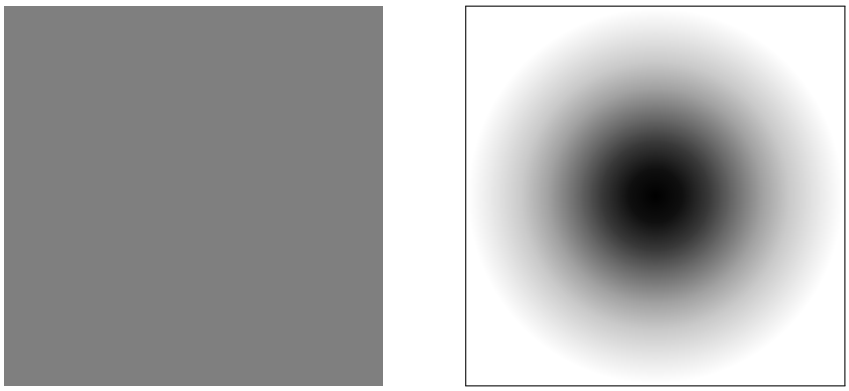


FIGURE 2.1: : Example for a mean/box filter on the left side and a Gaussian filter on the right side.

FIGURE 2.2: Examples of smoothing filters from left to right: Original image; Box filtered image; Gaussian filtered image. The mask size for both filters was 9x9.

The Gaussian function is usually used for weighting:

$$G_\sigma(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}} \tag{2.8}$$

In 2D case:

$$G_\sigma(x,y) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{2.9}$$

The integral of the Gaussian function is 1 and has the effect of preserving the result of the integral of the convolving of a Gaussian G with another function f:

$$\iint G(x,y) * f(x,y)\,dxdy = \iint f(x,y)\,dxdy \tag{2.10}$$

For the discrete case, a filter mask must be created that approximates the 2D Gaussian function. Although the Gaussian is defined over the entire domain $[-\infty,\infty]$, the function value quickly goes close to zero after a short distance, so the function can be truncated without losing much of the filter signal. A good distance to choose would be $[-4\sigma, 4\sigma]$, since $G(4\sigma)$ is close to zero. Using a shorter distance, such as 2σ, will result in strong mask boundaries and a boxy effect in the filtered image.

In the discrete case, normalization is simplified by dividing the computed unnormalized Gaussian values by the sum of all values in the mask. As a result, the sum of the mask is 1:

$$G[i,j] = e^{-\frac{i^2+j^2}{2\sigma^2}}; S = \sum_i \sum_j G[i,j]\,; G_n[i,j] = \frac{G}{S} \tag{2.11}$$

The mask size ks can be determined depending on the chosen σ for the Gaussian function's spread and the function's range of coverage r. To get good results, a sufficiently large range must be chosen, e.g. $r = [-4\sigma, 4\sigma]$ (coverage of the integral $> 99\%$). If $\sigma = \frac{1}{2}$ is set, a 5x5 mask is obtained with a center at $(0,0)$ and a range of $r = [-2,2]$. By defining the cover ratio variable cr (in this example, $cr = 4$), the kernel size can be determined as $ks = 2\lceil cr * \sigma \rceil + 1$, and the range as $r = [-cr\sigma, cr\sigma]$.
To simplify the Gaussian equations, set $\sigma = \frac{1}{\sqrt{2}}$ and introduce a normalization constant c with the default value $c = \frac{1}{\sqrt{\pi}}$. Then the Gaussian can be rewritten as a simplified equation:

$$G(x) = ce^{-x^2} \tag{2.12}$$

Discrete filters can be easily controlled by specifying a range of values, e.g., $\left[-\frac{4}{\sqrt{2}}, \frac{4}{\sqrt{2}}\right]$ and a filter size, e.g., 5, to reconstruct the example above. Then, for a given coverage cr, the kernel is determined as $ks = 2\left\lfloor \frac{cr}{\sqrt{2}} \right\rfloor + 1$ and the range as $r = \left[-\frac{cr}{\sqrt{2}}, \frac{cr}{\sqrt{2}}\right]$.

To control the spread of the simplified Gaussian filter (even with a constant σ), the cover ratio can be changed while the kernel size is fixed. Instead of using a variable covering rate, a sampling spacing can be used: $sp = \frac{cr}{\sqrt{2}\left\lfloor \frac{ks}{2} \right\rfloor}$. This means that $r = \left[-sp\left\lfloor \frac{ks}{2} \right\rfloor, sp\left\lfloor \frac{ks}{2} \right\rfloor\right]$. For a small spacing $(0 < sp < 1)$ the filter is spread, for a large spacing $(s > 1)$ the filter is compressed. The equivalent σ for a given sample spacing can be determined as $\sigma = \frac{1}{\sqrt{2}sp}$.

2.3 The Derivatives

To detect the intensity changes in images, the mathematical theorems of differential calculus [204] can be applied. The steepness of an intensity change can be analyzed by determining the derivatives of the intensity function. To locate the steepest change in intensity, one must find either the extrema within the first order derivative or the zero crossings within the second order derivative.

2.3.1 The Image Gradient

The image gradient represents the first derivative of an image (see Figure 2.3) by finding the partial derivatives for the two-dimensional intensity function $I(x,y)$:

$$I^{'}(x,y) = \nabla I(x,y) = \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \left[\frac{\partial I(x,y)}{\partial x}, \frac{\partial I(x,y)}{\partial y} \right]^t \tag{2.13}$$

Usually the gradient is defined as the magnitude m and the direction θ:

$$\begin{aligned} m &= \sqrt{g_x^2 + g_y^2} \\ \theta &= atan2\left(g_y, g_x\right) \end{aligned} \tag{2.14}$$

Intuitively interpreted, the gradient points in the direction of the steepest descent or ascent of intensity at each location (x,y) in the image. It maximizes the directional derivative. The magnitude reflects how quickly the intensity changes in the direction of steepest descent. The gradient magnitude is rotationally symmetric, or isotropic. Thus, it is independent of the orientation of the gradient vector. However, it is also a nonlinear operator that loses sign information on both sides of the edge (intensity change from light to dark or dark to light).

2.3.2 The Image Laplacian

The second derivative is defined as:

$$I^{''}(x,y) = \nabla^2 I(x,y) = \Delta I(x,y) = \begin{bmatrix} l_x \\ l_y \end{bmatrix} = \left[\frac{\partial^2 I(x,y)}{\partial x^2}, \frac{\partial^2 I(x,y)}{\partial y^2} \right]^t \tag{2.15}$$

The image Laplacian is isotropic and preserves the sign of intensity differences across edges. The zero crossings correspond to edges in the image, and the sign provides information about the intensity change. It is the lowest order linear combination of partial derivatives that is isotropic. The Laplacian can also be used to enhance contour effects in images:

$$I_c(x,y) = I(x,y) - \Delta I(x,y) \tag{2.16}$$

The most prominent derivative-based edge detectors are the Canny edge detector [31], which uses first-order Gaussian derivatives, and the Marr & Hildreth edge detector [147], which uses second-order Gaussian derivatives. These are discussed in more detail in the next chapter. An example is given in Figure 2.3.

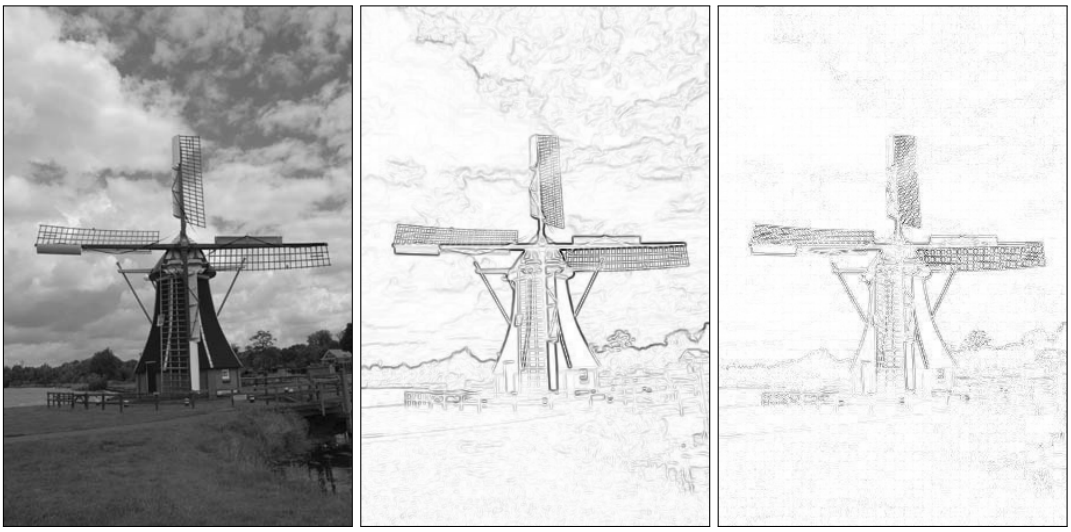


FIGURE 2.3: Derivative filter examples from left to right: Original image, Sobel filter and Laplacian filter. The mask size for both filters was 3x3. For the Laplacian image, the absolute filter response was used.

2.3.3 Approximate Image Derivatives with Linear Filters

Computing a derivative in the discrete case is a linear operation. The first derivative $f'(x)$ can be computed by convolving $f(x)$ with a function $h(v)$ that has a closely spaced positive and negative unit impulse. If the distance between the two pulses is defined as ε, then it is similar to the finite difference approximation for a derivative depending on ε:

$$\nabla f(x) \approx f(x) - f(x+\varepsilon) = f(x) * h(v) \tag{2.17}$$

The smaller ε, the better the approximation. The derivative approximation can easily be applied to an image intensity function:

$$\nabla I(x,y) = [I(x,y) - I(x+\varepsilon,y), I(x,y) - I(x,y+\varepsilon)]^t = I(x,y) * h(u,v) \tag{2.18}$$

2.3.4 Combine Smoothing and Derivatives

To reduce noise, smoothing is applied before differentiation, but since convolution is associative, it is also possible to write:

$$\nabla (I(x,y) * G(u,v)) = \nabla I(x,y) * G(u,v) = I(x,y) * \nabla G(u,v) \tag{2.19}$$

The derivative of a filtered image as a separable filter can be expressed as:

$$\nabla \mathrm{C}(x,y) = I(x,y) * \left(\nabla G_x(u) G_y(v) + G_x(u) \nabla G_y(v)\right) \tag{2.20}$$

Smoothing and derivation can be combined into a single filter mask that is used by many edge detectors. For example, the first derivative of the Gaussian is used by the Canny edge detection method [31]. Marr & Hildreth [147] used the second derivative of Gaussian for their approach, the Laplacian of Gaussian (LOG):

$$\nabla^2 (G * I) = \nabla^2 G * I = \Delta G * I \tag{2.21}$$

2.3.5 Effect on Edges

Considering a one-dimensional edge function $e(x)$ smoothed by a Gaussian function $G(u)$, the response to a step edge can be controlled by σ. The smaller the value of σ, the smaller the range of the response and the less blurred the localization, but also the less noise reduction. Thus, there is a trade-off between detection and localization: with a large σ, edges can be separated from noise at the expense of localization certainty and the loss of weak edges. With a small σ, the probability of detecting real edges decreases, while the probability of detecting a false edge increases. This problem can be solved by using a scale space.

2.4 Scale Space

A major problem with derivative edge detectors is the choice of an appropriate σ for image smoothing. The larger σ is chosen, the higher the probability of detecting a true edge, but the less detail edges are preserved in the images. In addition, the spatial localization of detected edges becomes worse. A scale space can be used to address the tradeoff between detectability and localization as a function of σ. Witkin [5] analyzed multiscale signals derived from Gaussian smoothing at different scales. He defined the scale space as a function over the domain of the intensity function $I(x,y)$ plus an additional dimension for the scale parameter, which is σ:

$$S(x,y,\sigma) = I(x,y) * G_\sigma(u,v) \tag{2.22}$$

The scale space has certain properties:

- The larger σ, the fewer edges can be detected
- The smaller σ, the more accurately the edges can be spatially localized
- If σ increases, existing edges may disappear, but never be added

The last property can be used to track edges across different scales in a scale tree. But it doesn't solve the problem of which scale to pick. One solution is to include the whole scale space in the processing and never pick a scale. Another approach is to pick a scale where detection is good, and then track the edge down the scale until its localization is also good. In this case, the edge may split into two disjoint edges as σ decreases, leaving the problem of which path to choose.

2.4.1 Differences of Gaussians (DOG)

The difference between images with different levels of smoothing will extract a subset of frequencies in the image, depending on the level of smoothing applied. The result is a bandpass filter. A common approach to implementing bandpass filters is to use images smoothed by a Gaussian (as seen in Figure 2.4, first row). Since a scale space tree represents the images in different scales of smoothing, subtracting the images of successive scales will result in images that contain a subset of the frequencies of the original image (as seen in Figure 2.4 second row):

$$I_{DOG} = S(x,y,\sigma_a) - S(x,y,\sigma_b), \quad \sigma_a < \sigma_b \tag{2.23}$$

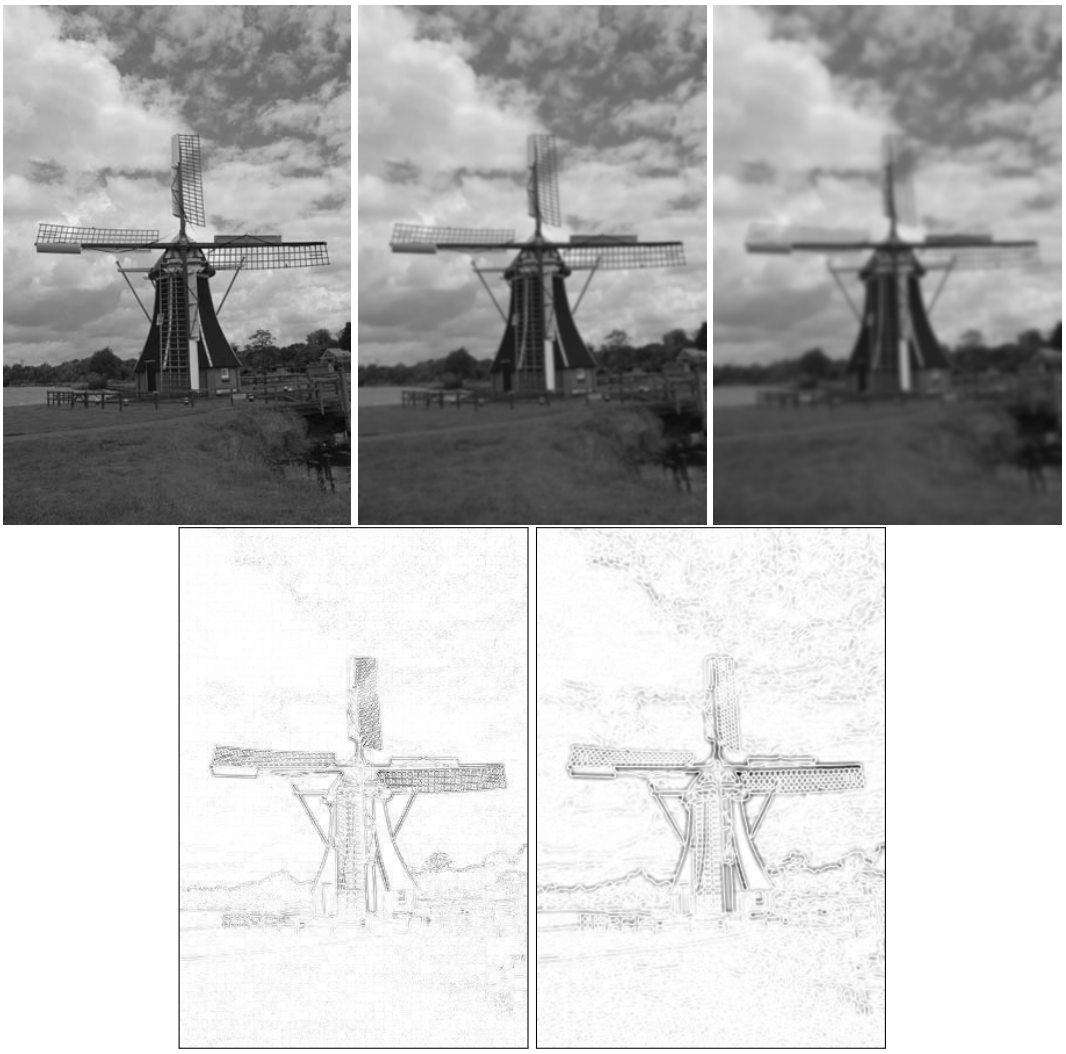


FIGURE 2.4: First row from left to right: original image, Gaussian smoothed image, stronger Gaussian smoothed image. Second row from left to right: absolute difference of the first two images in the first row, absolute difference of the second two images in the first row.

The differences of two Gaussians can also be used to approximate the second derivative operator LOG, since the LOG is also a bandpass filter:

$$LOG(x) = \Delta G(x) \approx DOG_{\sigma_a,\sigma_b}(x) = \frac{1}{\sqrt{2\pi\sigma_a^2}} e^{\frac{-x^2}{2\sigma_a^2}} - \frac{1}{\sqrt{2\pi\sigma_b^2}} e^{\frac{-x^2}{2\sigma_b^2}}, \quad \sigma_a < \sigma_b \tag{2.24}$$

The approximation holds only if the scales σ_a and σ_b are chosen within a certain ratio to each other: $\sigma_b/\sigma_a \approx 1.6$. The DOG is still a second derivative filter using different ratios. Ma et al. even suggest using a ratio of 5.68 to get better results [229].

2.4.2 Wavelets

A Scale space in this context can also be interpreted in terms of the wavelet theory. A Wavelet function must satisfy two properties:

- The integral of the function must be zero: $\int_{-\infty}^{\infty} h(x) = 0$
- A scaling condition must be fulfilled: $h_s(x) = \frac{1}{s} h(\frac{x}{s})$

Then the wavelet transform of the function f at the scale s is defined as:

$$W_s^h f(x) = f(x) * h_s(x) \tag{2.25}$$

The wavelet is computed by convolving the function f with the wavelet function h at some scale s.

The Gaussian used to smooth the images for the scale space satisfies the scaling property, where the scale factor s is represented by σ:

$$G_\sigma(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}} \tag{2.26}$$

The derivatives of the Gaussian also satisfy the zero-integral property and can be used as a wavelet function [145]. Thus, the differences of the smoothing scale space of $I(x,y)$, previously defined as I_{DOG}, can be interpreted as a wavelet transform. All edge detectors that use smoothing and differentiation with integral of zero can be interpreted as wavelet functions [130].

Wavelet transforms have the property of being an overly complete or highly redundant representation of a function. The dyadic wavelet transform, which involves smoothing only at the doubling of the scale (called octaves), is still a complete representation of a function:

$$Wf = (W_{2^i} f(x))_{i \in N},\ N = \{1, 2, 3, \dots\} \tag{2.27}$$

It is possible to reconstruct f from Wf. Even if only the filtered extremes or zero crossings are used, f can be completely reconstructed in many cases. So just the edges alone, obtained from different scales, capture most of the information of the original image.

A wavelet transform can also be used to analyze the discontinuity behavior of edges within a function by applying the notion of Lipschitz continuity. A generalized function $f(x)$ (which can include steps and deltas, and need not be continuous) is

uniformly Lipschitz of order α over the interval $[a,b]$ if there exists a constant $K>0$ such that for any $x_1, x_2 \in [a,b]$, the following holds:

$$|f(x_1)-f(x_2)| \leq K|x_1-x_2|^{\alpha},\ \alpha \in [-1,1] \tag{2.28}$$

By looking at the upper bound α_0 of all α, the continuity of $f(x)$ at x_0 can be interpreted:

- $\alpha = 1$: It is continuous and differentiable, the slope is bounded by some K and the magnitude $|f(x_1)-f(x_2)|$ is bounded by $K|x_1-x_2|$
- $\alpha = 0$: It is not continuous. The slope is not bounded by K, but the magnitude is bounded
- $\alpha = -1$: It is discontinuous and unbounded at x_0. The magnitude becomes arbitrarily large as the interval $[a,b]$ becomes arbitrarily small.

The Lipschitz theorem can be applied to wavelets. The function $f(x)$ is uniformly Lipschitz of order α over the interval $[a,b]$ if there exists a constant $K > 0$ such that, for any $x \in [a,b]$, the wavelet transform satisfies:

$$W_{2^i} f(x) \leq K(2^i)^{\alpha} \tag{2.29}$$

This means that if $\alpha > 0$, the magnitude of the wavelet transform at x_0 decreases as the scale decreases. If $\alpha = 0$, the magnitude at x_0 is independent of the scale. And if $\alpha < 0$, the magnitude at x_0 will decrease as the scale increases. The information about α can help classify the type of an edge as a differentiable, step, or delta function based on the wavelet magnitude of the different scales.

For the discrete case, this information can be used to distinguish between continuous and discontinuous edges over scale. By analyzing the magnitude behavior over different scales, assumptions can also be made about the edge type:

- If the magnitude of a peak in the smoothed derivative decreases with decreasing scale, it corresponds to a roof or peak edge.
- If it stays the same, it corresponds to a step edge.
- If it becomes larger, it could be interpreted as a ramp or continuous edge that becomes sharper with decreasing scale.

By analyzing these properties, wavelet transforms facilitate robust edge detection and feature extraction, making them fundamental in image processing and computer vision applications [151].

Chapter 3

Image Preprocessing

To improve the edge extraction results, some optimization steps can be applied beforehand. In some cases, it is even necessary to apply some preprocessing, e.g. if a special input is required for an edge detector, such as converting a color image into a grayscale image. This leads to the discussion of color information in general. How important is color information for edge detection and what can and should be considered.

After discussing the color information for edge detection, this chapter also examines how to improve edge detection in terms of precision and artifacts caused by noise. Higher precision is needed to improve the localization of edge pixels, while image denoising is used to reduce artifacts in the image that could lead to false edge responses. These steps are already built into some edge detection methods, but in order to develop a better understanding of them and to be able to improve even primitive approaches, they are also introduced in this chapter.

3.1 Information of Color

According to the study by Novak et al. [162], about 90% of edge pixels are equivalent, estimated from grayscale and color images. Own tests based on the Berkeley image dataset (BSDS500) [7], give similar results (see subsection 3.4.1 for more details). For most applications this 10% of lost edge responses for grayscale images is not significant and can be ignored. In special cases, however, the color information may be essential for post-processing.

For some categories of edge detectors, the color space can be easily incorporated without loss of efficiency. A good example is the morphological operators [51] introduced in subsection 4.4.1. For other methods, this step often means more complex processing and an additional fusion step. For a comprehensive analysis, see the work of Zhu et al. [239].

For classical gradient-based edge detection operators, three categories can be found for color images [186]:

1. Output fusion methods: Edge detection is processed independently for each color channel and merged in a final fusion step. Different approaches to data fusion have been developed and various color models have been considered [159, 33, 188].
2. Multidimensional gradient methods: These methods combine gradients processed separately for each color channel into an edge response. Zenzo [43]

discussed various approaches to gradient fusion using the norm or the tensor product. Other methods can be found in [215, 190].

3. Vector methods: In this case, the vector nature of the color is preserved throughout the processing. Several approaches exist, as described in [143, 214]. A vector-like approach was developed by Ruzon et al. [187], based on quantized color signatures and a perceptual color distance computed by the Earth Mover's Distance [184].

The use of color information for edge extraction in real-time applications should be carefully considered, as in most cases the additional information is not significant, but requires much more processing time.

3.2 Upscaling

To improve the precision of edge responses which later allow to provide more accurate edge location, the input image can be upscaled beforehand. However, this has the disadvantage that much more data has to be processed when determining the edge responses, which is a major technical disadvantage, especially in terms of runtime. Therefore, this topic will not be discussed any further here. More efficient approaches are discussed in section 6.3 using interpolation methods.

3.3 Image Denoising

Images taken with a camera contain noise, even if it is not visible to the naked eye. It is caused by the image sensor itself , depending on settings like sensor sensitivity [68], but also scratches, dead pixels, compression, etc. can add noise to images .

The noise produced by the sensor is often uniformly distributed and interpreted as zero mean noise (e.g. white Gaussian noise) [89]. It can be reduced with linear filters, e.g. a Gaussian smoothing filter. The other sources of noise mentioned usually lead to more selective inference in the image and are called impulse noise [68, Chapter 5]. Impulse noise is often poorly eliminated by linear filters. However, nonlinear filters such as the median filter [96] can be used to reduce this image noise more efficiently.

3.3.1 Gaussian Smoothing

The classic preprocessing for edge detection is to apply Gaussian smoothing to reduce noise and other artifacts caused by high frequencies. Many edge detection methods already have smoothing or noise reduction built in, so they can be applied without smoothing. However, depending on the type or amount of noise in an image, other smoothing or denoising methods may still improve edge detection results.

If the image contains a lot of noise, strong Gaussian smoothing will help to suppress more noise, but at the same time image details will be lost and strong edges will be blurred, resulting in undetected or poorly localized edges. To overcome this problem, more structure preserving filters can be applied for denoising. There are many nonlinear methods that can effectively and efficiently suppress noise in images [28]. A simple but quite fast method is the median filter [96]. It replaces each pixel in the image with the median of its neighbors. For weak and moderate zero-mean noise, and for impulsive noise, the median outperforms the Gaussian blur due to

better contour preservation. For strong high frequency noise, its performance is not much better [8].

3.3.2 Bilateral Filters

Another approach is the use of bilateral filters. It was introduced by Tomasi et al. [209] and is a nonlinear, edge preserving and noise reducing smoothing filter. Each pixel in an image is replaced by a weighted average of the values of nearby pixels. Typically, the weighting is based on a Gaussian distribution. But instead of simply depending on the Euclidean distance of the neighboring pixels, it is also influenced by radiometric differences such as the intensity difference between the two pixels. As a result, strong edges are preserved while noise and texture are smoothed.

Bilateral filters have some problems such as the staircase effect, which can lead to homogeneous areas of intensity in images, giving them a cartoon-like look. Since they can also be used to enhance details or do other image manipulation, another problem can occur by not preserving the gradients, leading to gradient reversals. This leads to false edges. A more efficient approach that overcomes this problem is the guided image filtering of He et al. [85].

3.3.3 Non-local Mean

Another approach is the non-local mean denoising introduced by Buades et al. [27]. Instead of computing a mean for a pixel based on its local neighborhood, the filter takes a mean based on all pixels in the image, weighted by how similar these pixels are to the target pixel. This results in less loss of detail compared to a local mean method.

3.3.4 Other Methods

There are other denoising methods such as total variation denoising [185], anisotropic diffusion [14] or wavelet transform [60]. However, these are not discussed further as there is no implementation available for a closer look or comparison.

3.4 Results and Discussion

So far, the importance of color information has been discussed and some preprocessing methods useful for edge detection have been introduced. This part of the chapter will dig a little deeper by analyzing a small experiment about color information in images and providing a comparison of the denoising methods available in the OpenCV library.

3.4.1 Significance of Color Information

To measure the information gain for natural color images compared to grayscale images, a set of 500 natural color images from the BSDS500 Berkeley dataset [7] are used. For grayscale images, the color images were converted using the OpenCV cvtColor method and then the Canny edge detector was applied. For color images, Canny was applied to each individual channel and the resulting edge maps were then merged into the final result (see Figure 3.1). Since this approach amplifies noise and creates thick edges (multiple NMS with similar but not identical results), the difference cannot be measured simply by comparing the number of resulting

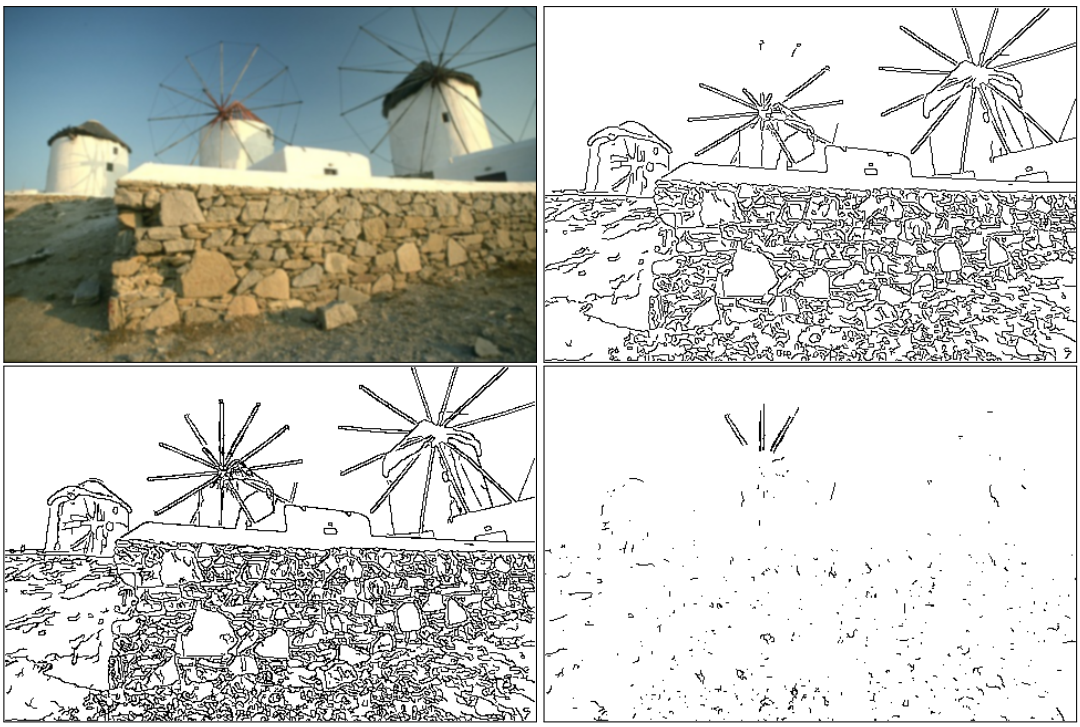


FIGURE 3.1: Top left: Example color image; Top right: Canny of grayscale image; Bottom left: Canny of color image; Bottom right: Difference of canny edge images

Blur	Lower Threshold	Upper Threshold	Dilation	Information Gain
0.600	0.004	0.012	3.000	12.595
1.000	0.004	0.012	3.000	13.439
0.600	0.010	0.030	3.000	11.592
1.000	0.010	0.030	3.000	13.193
0.600	0.030	0.060	3.000	11.152
1.000	0.030	0.060	3.000	15.919
0.600	0.004	0.012	5.000	6.558
1.000	0.004	0.012	5.000	6.932
0.600	0.010	0.030	5.000	5.698
1.000	0.010	0.030	5.000	7.006
0.600	0.030	0.060	5.000	5.561
1.000	0.030	0.060	5.000	10.031

TABLE 3.1: This table shows the parameter settings for the different test cases and the resulting average information gain for each test case based on Canny.

edge pixels. Instead, the resulting gray images are additionally dilated and used to mask the results from the color images, yielding the final difference. All pixels of the color images not removed by the mask (and the operation of the inverse of the mask) are counted and used to measure the difference. Several variations of the parameters were used to obtain a final average information gain for the color images (see Table 3.1).

A simple 3x3 or 5x5 mask was used as the dilation structuring element. While the 3x3 dilation mask filters less information, preserving additional color pixel data near mutual data, it also leaks false edge data. The 5x5 mask, on the other hand, is better at filtering noise from mutual edges, but also filters relevant data. The final result was estimated by the average of the computed results: 9.973%.

As shown in Figure 3.1, in most regions the edge image difference doesn't show significant occurrences. In most cases, only short clutter edge support regions remain. The most significant exception is the wheel of the first windmill. The grayscale Canny misses a large part of it, while the color version detects it with the same thresholds. This could be fixed for the grayscale canny by tweaking the thresholds a bit, but it still shows that color can be important for this task.

Blur	Lower Threshold	Upper Threshold	Dilation	Information Gain
0.600	0.004	0.012	0.000	18.420
1.000	0.004	0.012	0.000	20.431
0.600	0.010	0.030	0.000	14.639
1.000	0.010	0.030	0.000	16.824
0.600	0.030	0.060	0.000	13.063
1.000	0.030	0.060	0.000	15.314
0.600	0.004	0.012	3.000	4.096
1.000	0.004	0.012	3.000	3.813
0.600	0.010	0.030	3.000	2.915
1.000	0.010	0.030	3.000	3.512
0.600	0.030	0.060	3.000	3.651
1.000	0.030	0.060	3.000	5.099

Table 3.2: This table shows the parameter settings for the different test cases and the resulting average information gain for each test case based on RCMG.

A similar test was performed with the RCMG method applied to color and gray images. An average information gain of 10.148% was obtained, further confirming the previous assumption. A dilation of 0 and 3 was used (see Table 3.2), since RCMG is less prone to produce thick lines (non maxima suppression is applied only once).

The current values of the experiment lack confidence as they seem to be influenced by the parameters chosen. They require further investigation to provide more reliable results, and additional analysis is required to provide more consistent and meaningful results.

Regarding the Canny edge detection approach, the default RGB to grayscale conversion applies human perception-based weighting to the color channels. In particular,

the conversion formula of OpenCV uses the following weighting[1]:

$$Y = 0.299 * R + 0.587 * G + 0.114 * B \tag{3.1}$$

It gives much more weight to the red and green channels, while the blue channel contributes less to the grayscale representation. This perceptual bias most likely explains the observed loss of edge responses in the windmill example (see Figure 3.1), where the reduced contribution of the blue channel may result in a weaker edge response for the grayscale variant. Thus, the information is not necessarily lost in the image and edge responses, but in the process of extracting the final edge pixels. As mentioned above, this could be compensated by adjusting the thresholds and still be much more efficient than the RGB variant.

3.4.2 Denoising

Method	N10	N20	N30	N40	N50	Time
none	0.031	0.061	0.091	0.118	0.144	0.000
box 3	0.021	0.029	0.038	0.047	0.056	0.465
box 5	0.027	0.031	0.036	0.041	0.047	0.464
box 7	0.033	0.035	0.038	0.042	0.047	0.466
box 9	0.037	0.039	0.042	0.045	0.049	0.470
gauss 3	0.019	0.029	0.039	0.049	0.059	0.315
gauss 5	0.021	0.027	0.034	0.041	0.049	0.424
gauss 7	0.024	0.028	0.033	0.039	0.045	0.770
gauss 9	0.027	0.030	0.034	0.038	0.043	0.958
median 3	0.020	0.032	0.044	0.056	0.067	0.162
median 5	0.023	0.030	0.037	0.043	0.050	0.566
median 7	0.026	0.032	0.036	0.041	0.046	14.246
median 9	0.030	0.034	0.038	0.042	0.045	15.369
bilat 3	0.018	0.032	0.051	0.075	0.100	3.577
bilat 5	0.017	0.026	0.041	0.061	0.085	7.287
bilat 7	0.019	0.025	0.037	0.057	0.080	14.827
bilat 9	0.021	0.025	0.037	0.055	0.078	24.158
nlmean 5	0.030	0.061	0.091	0.118	0.144	422.463
nlmean 10	0.014	0.056	0.090	0.118	0.144	412.214
nlmean 15	0.016	0.026	0.078	0.116	0.144	413.625
nlmean 20	0.020	0.022	0.039	0.096	0.138	414.395

Table 3.3: Includes the average absolute error after denoising and the average processing time. Five different noise levels were used (column N10-N50).

To evaluate the denoising methods built into OpenCV (see section 3.3), a simple abstraction framework was created to encapsulate the different filters into a common interface so that they can be used within a test pipeline. Given an image set (MDB-Q dataset), the test pipeline estimates the average absolute error for all images after the applied denoising filter and the average filtering time. The error is computed as the average absolute difference between the original image (converted to grayscale) and the filtered image, which was previously imposed with Gaussian noise. The results

[1] taken from `https://docs.opencv.org/3.4/de/d25/imgproc_color_conversions.html`

are displayed in Table 3.3.

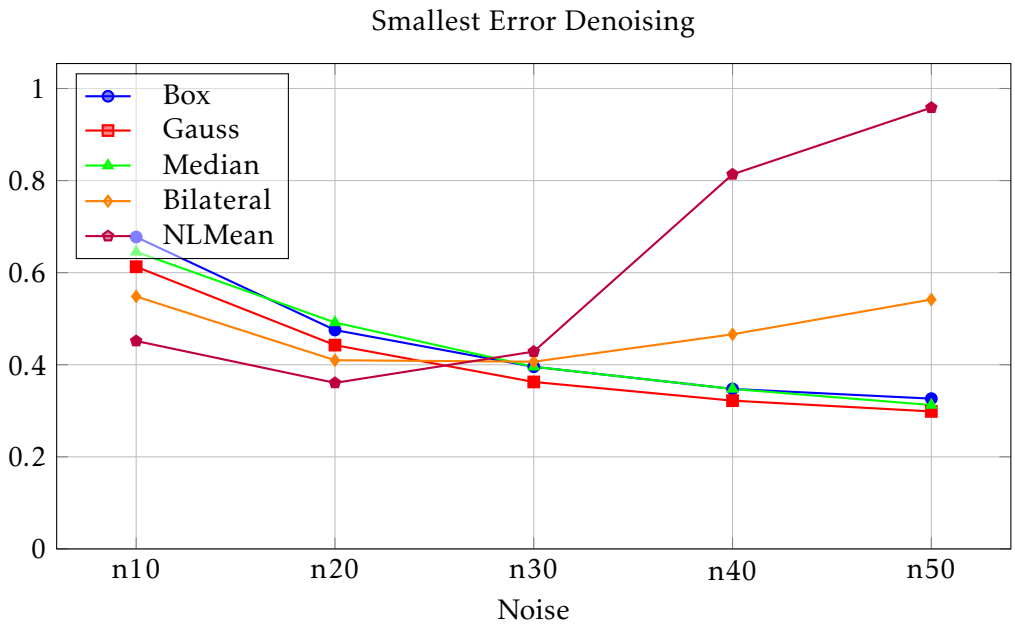


FIGURE 3.2: This chart illustrates the minimum error values for each method of the given parameter set. The errors are expressed relative to the real error (first row in Table 3.3): 1 no improvement, 0 perfect denoising.

Multiple methods were used with four different parameter settings. The number in the Method column indicates the parameters used. For the Box and Median filters, there is only one parameter, the kernel size. For the Gaussian filter, an appropriate sigma is estimated from the given kernel size. The additional parameters for the bilateral filter (sigma for color space and sigma for coordinate space) are set to a fixed value of 50. The non-local mean method uses a fixed template window of 7 pixels and a search window size of 21 pixels. Only the filter strength has been changed.

In Figure 3.2, the smallest error for each method at different noise levels is shown. It shows that the more complex methods, such as the non-local mean filter or the bilateral filter, outperform the simple methods for weak and moderate noise. For high noise, the complex filters need better parameter tuning to get good results, while the simple filters continuously improve the results by simply blurring the noise more.

The Figure 3.3 shows the execution time for the denoising methods. The non-local mean methods have been excluded because of their high and constant runtime. The constant runtime can be explained by the fixed window size parameters. For the other methods, the graph nicely reflects the complexity of the filtering. The simple methods have a fairly flat increase in runtime with increasing filter size, while the complex bilateral filtering is steep and quadratic. It seems that OpenCV implements a highly efficient variant of the median filter only for small kernel sizes. This would explain the jump in the graph.

The Figure 3.4 shows an example set of the filtered images. It can be observed that the complex methods are better at preserving edge information. In Figure 3.5 the resulting edge maps are shown. It shows how the added noise affects the edge

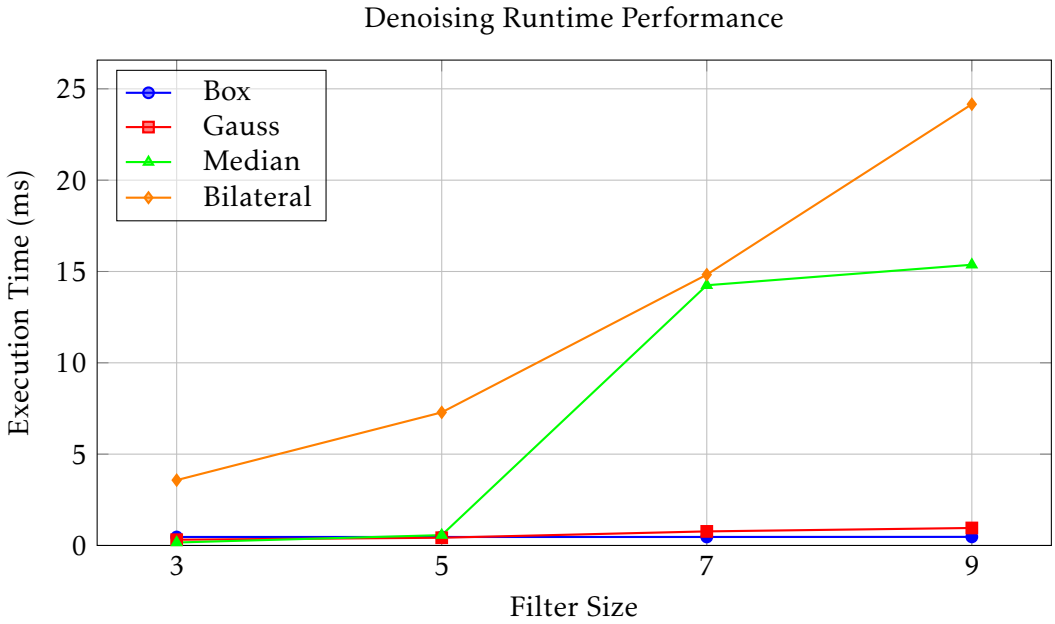


FIGURE 3.3: Illustrates the execution time (ms) for the denoising filtering methods with increasing filter size (see section 5.1 for hardware setup).

response. By applying the denoising methods, the visibility of the edge response is improved in all cases, as the outlines of the objects in the image stand out more clearly from the clutter.

Similar to upscaling, this can be used as a pre-processing step to improve the data before feeding it to the edge detector. However, it cannot be said that this will generally significantly improve the results. Depending on the use case and time constraints, it must be evaluated whether such a step is beneficial, not only because many detectors already include denoising in their processing, but also whether the improved results are worth the additional runtime cost.

FIGURE 3.4: Example images to visualize the denoising results. A Gaussian noise with an intensity of 20 was applied before denoising.

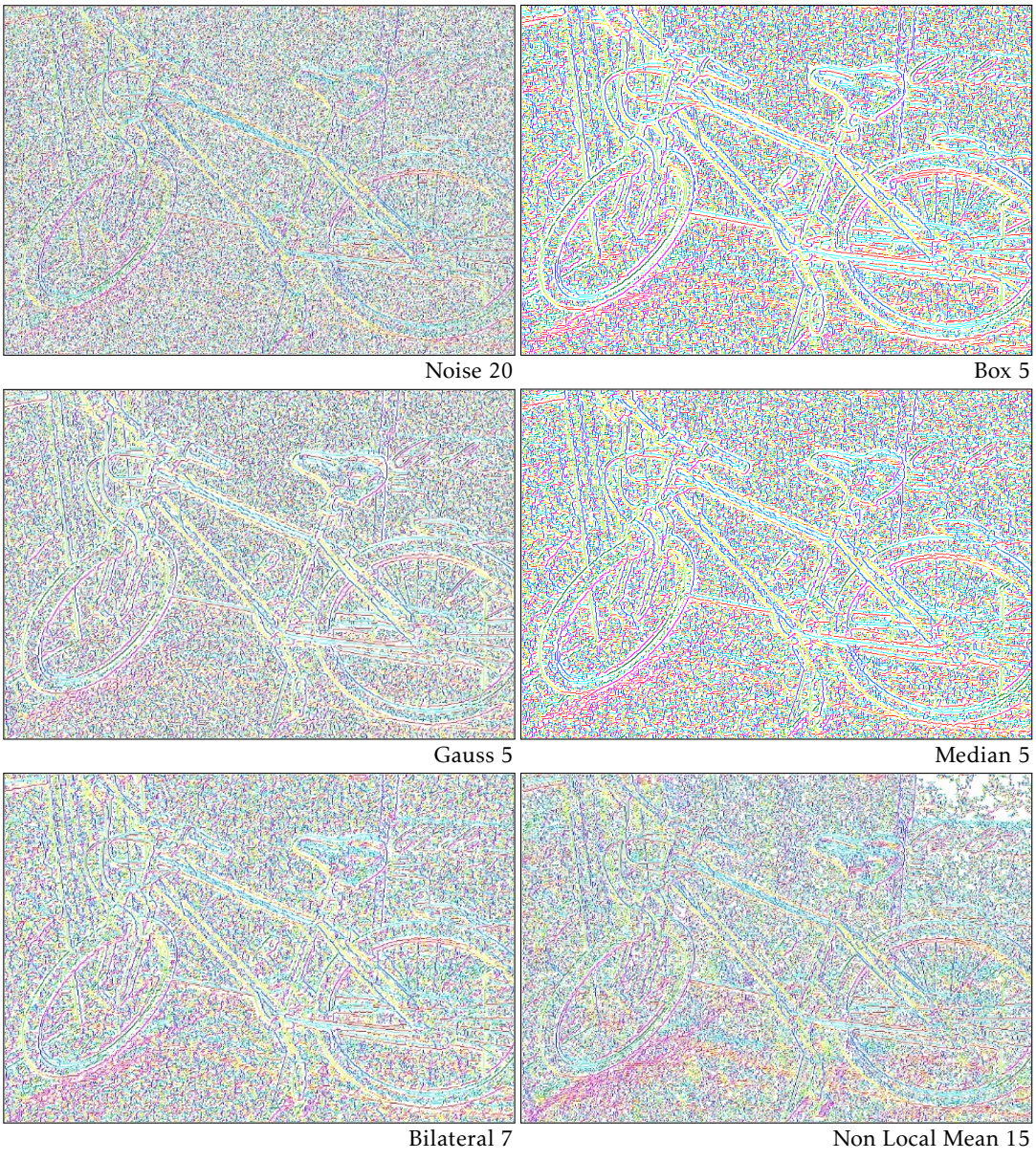


FIGURE 3.5: Colorized edge maps (by orientation) to visualize the edge results after denoising.

Chapter 4

Edge Region Detectors

Since the question of "what are edges in images" has been answered, and some basic concepts of image processing have been introduced, this chapter will discuss the methods for edge detection in detail.

For real-time edge detection, local operators are preferred because they outperform global methods in terms of runtime. Therefore, this chapter focuses on local edge detectors and only touches on the global approaches. Their disadvantage is that only a small neighborhood of adjacent image intensities is considered to determine whether an image point belongs to an edge or not. On the other hand, these approaches can be efficiently processed by parallel implementations. The global aspects are partially covered in chapter 7 by finding connected components within the edge regions.

4.1 Derivative Filters

One of the most common approaches to finding edge regions is to use derivative filters. section 2.3 already introduced derivatives in continuous form. Since in image processing an image is discrete, the derivatives have to be approximated. This can be done by simply computing the differences of the discrete intensity values of an image [203]:

$$\begin{aligned} \nabla_x I(x,y) &= I(x,y) - I(x-1,y), \\ \nabla_y I(x,y) &= I(x,y) - I(x,y-1) \end{aligned} \tag{4.1}$$

This can also be done by convolving the image with a mask of differences. The Roberts operator uses two 2x2 diagonal masks:

$$\begin{aligned} r_1 = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix},\ r_2 = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}, \\ \theta = atan2(I * h_2, I * h_1) + \frac{\pi}{4} \end{aligned} \tag{4.2}$$

The Prewitt [174] operator uses a simple 3x3 difference mask:

$$p_1 = \frac{1}{3}\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix},\ p_2 = \frac{1}{3}\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \tag{4.3}$$

They can also be expressed as separable filter pairs a and b:

$$\begin{gathered} p_1(x,y) = a(x)\,b(y), \quad p_2(x,y) = b(x)\,a(y) = p_1^t \\ a = \frac{1}{3}[1\ 1\ 1],\ b = [-1\ 0\ 1], \end{gathered} \tag{4.4}$$

Instead of calculating only the vertical and horizontal mask, other directions can be included, such as the diagonal case:

$$p_3 = \frac{1}{3}\begin{bmatrix} 0 & 1 & 1 \\ -1 & 0 & 1 \\ -1 & -1 & 0 \end{bmatrix},\ p_4 = \frac{1}{3}\begin{bmatrix} -1 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \tag{4.5}$$

Even more directions can be computed, but in this case larger and smoother filter masks are required. The magnitude and direction are then determined by selecting the mask with the maximum response. For more details, see subsection 4.2.3

The filter mask for a second derivative, the Laplacian [61] can be defined in 1D as:

$$\nabla^2 = [1\ -2\ 1] \tag{4.6}$$

A 2D Laplacian is then defined as:

$$\nabla^2 = \begin{bmatrix} 0 & 0 & 0 \\ 1 & -2 & 1 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 1 & 0 \\ 0 & -2 & 0 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{4.7}$$

Additionally, the response to 45° rotated edges can be computed, and combined with the horizontal and vertical masks:

$$\nabla^2 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{4.8}$$

The Laplacian mask is isotropic and requires only one convolution to compute the derivative. However, it is not separable and the edge direction is not determined.

Since derivative filters are very sensitive to noise, filter masks with additional smoothing are more effective for edge extraction. One of the most popular filter masks is the Sobel operator:

$$so_1 = \frac{1}{4}\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix},\ so_2 = \frac{1}{4}\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{4.9}$$

The Sobel operator is separable and contains a slight Gaussian smoothing [199]. An optimization of this filter mask was done by Scharr [191]. Since the gradient orientation is an important information for many image processing tasks, he wanted to improve the estimation of the gradient orientation. He optimized his gradient

operator in the Fourier domain and proposed the following filter masks:

$$sa_1 = \frac{1}{16}\begin{bmatrix} -3 & 0 & 3 \\ -10 & 0 & 10 \\ -3 & 0 & 3 \end{bmatrix}, \; sa_2 = \frac{1}{16}\begin{bmatrix} -3 & -10 & -3 \\ 0 & 0 & 0 \\ 3 & 10 & 3 \end{bmatrix} \tag{4.10}$$

The Scharr operator is also separable and incorporates Gaussian smoothing similar to Sobel, but with a different standard deviation.

Authors like Canny tried to find optimal smoothing filters with respect to some edge detection criteria. They used separable filters to implement them efficiently.

4.1.1 Canny Filter

Canny used first derivative filters for his approach. He focused on finding optimal step edge detectors in images with zero mean white noise and gave three criteria for detector optimization [31]:

- **Good detection:** low probability of missing real edges and low probability of detecting false edges.
- **Good localization:** detected edges should be as close as possible to the true locations of the edges.
- **Minimal response:** a single edge should ideally produce a single response in the output.

From these criteria, Canny derived a general solution to a second-order linear homogeneous differential equation with constant coefficients:

$$h(x) = a_1 e^{ax}\cos(\omega x) + a_2 e^{ax}\sin(\omega x) + a_3 e^{-ax}\cos(\omega x) + a_4 e^{-ax}\sin(\omega x) + c \tag{4.11}$$

The solution is antisymmetric with respect to $x = 0$ and is defined for $-\omega < x \leq 0$. For $x > 0$, the function is obtained by reflection: $h(x) = -h(-x)$.
For simplicity and computational efficiency, Canny approximated this solution using the first derivative of a Gaussian. The Gaussian function is defined as:

$$G(x) = e^{-\frac{x^2}{2\sigma^2}} \tag{4.12}$$

and its first derivative is:

$$G'(x) = -\frac{x}{\sigma^2}\, e^{-\frac{x^2}{2\sigma^2}} \tag{4.13}$$

The one-dimensional filters G and G' can be applied as a separable filter pair to compute partial derivatives in the x- or y-direction of an image, in a similar fashion to operators such as Sobel or Scharr.

4.1.2 Canny-Based Filter Extensions

Following Canny's criteria, several authors have developed related edge detectors. Deriche [39] proposed a smoothing filter that approximates Canny's differential equation using a Infinite Impulse Response (IIR) filter instead of a Gaussian Finite Impulse Response (FIR) filter (e.g., convolution kernels such as Sobel or Scharr) [140].

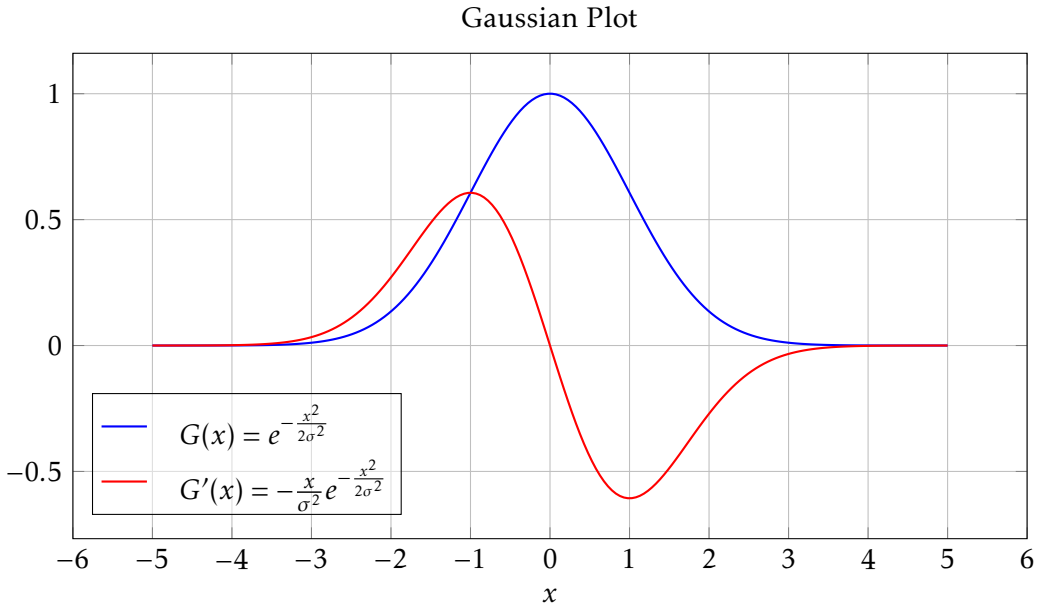


FIGURE 4.1: Plot of Gaussian function and its first derivative with $\sigma = 1$.

Deriche's IIR filter is defined as:

$$\begin{aligned} h(x) &= c\,(\alpha|x|+1)\,e^{-\alpha|x|}, \\ h'(x) &= -c'xe^{-\alpha|x|}, \\ c = \frac{(1-e^{-\alpha})^2}{1+2\alpha e^{-\alpha}-e^{-2\alpha}}, &\quad c' = \frac{(1-e^{-\alpha})^2}{e^{-\alpha}} \end{aligned} \tag{4.14}$$

Shen et al. [196] applied only the first two of Canny's optimization criteria and obtained a solution similar to Canny's. Their approximation can be expressed as:

$$\begin{aligned} h(x) &= ce^{-\alpha|x|}, \\ h'(x) &= \begin{cases} c'e^{-\alpha|x|}, & x \geq 0, \\ -c'e^{-\alpha|x|}, & \text{otherwise} \end{cases} \\ c = \frac{1-e^{-\alpha}}{1+e^{-\alpha}}, &\quad c' = 1-e^{-\alpha} \end{aligned} \tag{4.15}$$

Here, c and c' are normalization constants, while α determines the filter width ($\sim \sigma^{-1}$). A smaller α produces a wider filter with stronger smoothing and less precise localization.

Lanser et al. [119] addressed the problem that the IIR filters of Deriche and Shen et al. cause distortion in edge amplitudes depending on their orientation, and proposed a modified detection procedure to eliminate this effect.

A generalization of the Deriche filter for noisy edge environments was presented by Bourennane et al. [22], which also results in an IIR implementation.

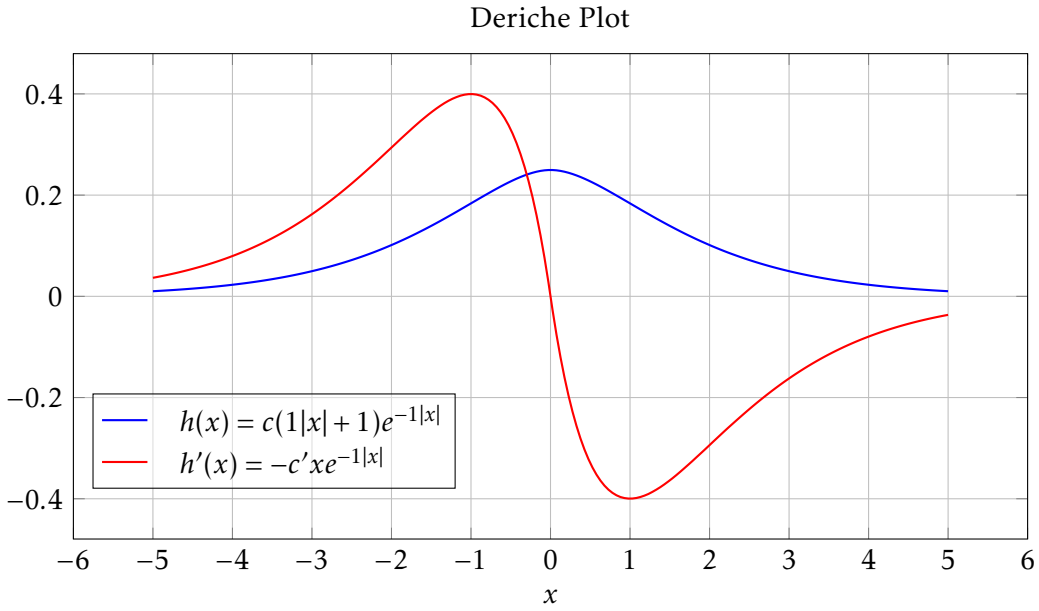


FIGURE 4.2: Plot of Deriche function and its first derivative with $\alpha = 1$.

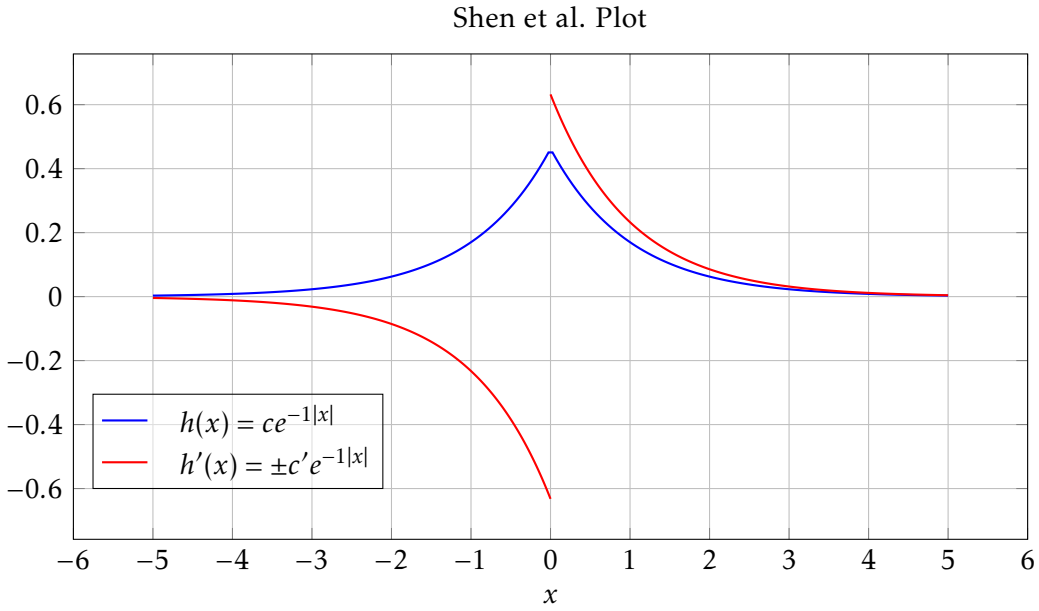


FIGURE 4.3: Plot of Shen et al. function and its first derivative with $\alpha = 1$.

4.1.3 Marr & Hildreth Filter

Inspired by models of human visual processing, Marr and Hildreth [147] proposed an edge detection method based on the Laplacian of Gaussian (LoG). The LoG operator applies Gaussian smoothing to an image, followed by the Laplacian, to highlight regions of rapid intensity change. In its simplified, unnormalized form (assuming $\sigma = \frac{1}{\sqrt{2}}$), the Gaussian can be expressed as:

$$h(x,y) = c\,e^{-(x^2+y^2)} \tag{4.16}$$

where c is a scaling constant.
Introducing the polar coordinate substitution $r^2 = x^2 + y^2$ yields:

$$\begin{aligned} h(r) &= ce^{-r^2}, \\ h'(r) &= \nabla h(r) = -2rc\,e^{-r^2}, \\ h''(r) &= \Delta h(r) = \left(4r^2 - 2\right) c\,e^{-r^2} \end{aligned} \tag{4.17}$$

Rewriting the second derivative in Cartesian coordinates by substituting $r^2 = x^2 + y^2$ gives:

$$h''(x,y) = \Delta h(x,y) = \left(4x^2 + 4y^2 - 2\right) c\,e^{-(x^2+y^2)} \tag{4.18}$$

The LoG shares the main properties of the Laplacian operator: it is isotropic, non-separable, does not encode edge direction, and is more sensitive to noise than first derivative operators. Edges are identified at zero crossings in the convolved image (see subsection 6.2.3).

4.1.4 Normalization and DC Response in Derivative Filters

Both first and second derivative filters, such as those mentioned in the last sections, are theoretically designed to have a zero DC response. This means, the sum of all filter coefficients should be exactly zero:

$$\sum_x \sum_y h(x,y) = 0 \tag{4.19}$$

This property ensures that constant regions in the input image produce a zero output, which is fundamental for derivative operators.

In discrete FIR implementations, the finite kernel size and sampling effects can cause the DC response to deviate from zero. If uncorrected, this introduces a bias toward low-frequency components and can create visible artifacts in the output.

To avoid such issues, the kernel coefficients are scaled or adjusted to enforce zero DC. For Gaussian derivative filters, this often means adjusting the normalization constant (e.g., c or c' in subsection 4.1.2) so that the discrete sum is exactly zero. In LoG filters, the same principle applies to the second derivative kernel (subsection 4.1.3). If the desired kernel size does not naturally yield zero DC, a post-scaling step is required to satisfy this condition (see also section A.1).

4.1.5 Local Line Fitting

Another approach to estimating the derivative of an image is to use line fitting. A line is fitted to a local pattern around each pixel, and then the derivative is calculated analytically from the fitting function [34]. The fitting can be done by polynomials or cubic splines. Shen shows in his paper [125] that local line fitting is related to local image moments. The moments can be computed quite efficiently compared to the line fitting.

4.1.6 Conclusion

Differential methods can be computed very efficiently and are used in most cases in combination with the edge extraction pipeline proposed by Canny. A major drawback is the inferior localization of edge pixels when the signal has to be heavily smoothed due to noise. Another problem is that these methods do not distinguish between edges caused by texture and edges caused by region or object boundaries. This can lead to undesired behavior because it does not correspond to human perception.

4.2 Local Energy and Phase Congruency

Instead of using feature detectors in the spatial domain, approaches to feature detection in the frequency domain have been investigated. Using quadrature filters, Morrone et al. [153] developed a model of feature perception called the local energy model. The model assumes that feature points can be found in an image where the Fourier components are maximally in phase.

4.2.1 Quadrature Filter

Quadrature filters are used in many biological systems. Banks of quadrature filter pairs are tuned to different spatial frequencies to obtain localized frequency information [58]. A quadrature filter pair consists of two filters with the same frequency response, but one of them is phase-shifted by 90°. They are expressed as a complex filter whose real part is related to the imaginary part by the phase shift. A common quadrature filter is the Gabor filter [38]:

$$G(x) = e^{-\frac{x^2}{2\sigma^2}} e^{i(\frac{2\pi x}{\gamma})} \tag{4.20}$$

The function represents a sine wave (second term) modulated by a Gaussian envelope (first term). The second term is complex and represents a cosine wave (real part) that is evenly symmetric about zero and a sine wave (imaginary part) that is oddly symmetric about zero with a phase shift of 90°. The wavelength is defined by γ. For other functions, such as derivatives of Gaussians, the phase shift can be determined using a Hilbert transform [101].

For image processing, 2D quadrature filters can be used to estimate local energy in images. The Hilbert transform is applied along an axis through the origin of the 2D filter mask. An example of how to create a quadrature filter pair without the Hilbert transform is to use a 2D Gabor function:

$$G(x, y) = e^{-x^2+y^2} e^{i\frac{\pi}{2}x} \tag{4.21}$$

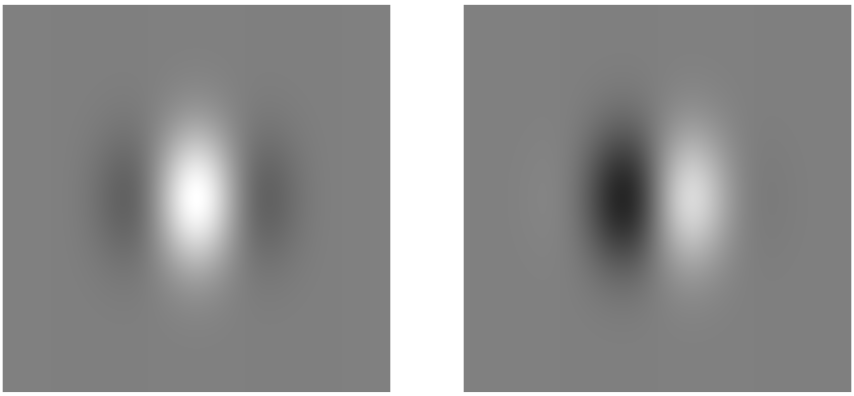


Figure 4.4: Cosine (even) and sine (odd) wave modulated by a Gaussian.

This Gabor function is oriented in the vertical direction, as shown in Figure 4.4 Cosine (even symmetry) and sine (odd symmetry) waves modulated by a Gaussian. The even-symmetric real part is sensitive to oriented roof edges or lines, like second derivative operators. The odd-symmetric imaginary part is sensitive to oriented step edges, like first derivative operators. Other choices for even and odd quadrature filter pairs are derivatives of Gaussians or logarithmic Gabor functions. See [21] for an overview and comparison of possible quadrature pair filters.
The result is two oriented filter kernels (in the example, the orientation is $\theta = 0$):

$$f^{\theta}(x,y) = real(G(x,y)), \;\; h^{\theta} = imag(G(x,y)) \tag{4.22}$$

The responses for the filters are computed by convoluting the image with the even and odd filter kernels:

$$e^{\theta} = f^{\theta} * I, \; o^{\theta} = h^{\theta} * I \tag{4.23}$$

Combining the responses of the even and odd filters gives a phase-independent measure of the orientation strength or local energy:

$$E^{\theta}(x,y) = \sqrt{e^{\theta}(x,y)^2 + o^{\theta}(x,y)^2} \tag{4.24}$$

The phase is given as:

$$\varphi^{\theta}(x,y) = atan2(o^{\theta}(x,y), e^{\theta}(x,y)) \tag{4.25}$$

Using this local energy model, not only step and roof edges can be correctly detected, but also transitions between them. Using first derivative operators like Sobel will result in multiple detections depending on the scale used. In Figure 4.5 a transition between a line and a step edge is shown. The Sobel filter result on the left in Figure 4.6 and Figure 4.7 shows the problem of multiple responses of a line by simple derivative operators, while this problem doesn't occur in the square response (right side in the figures).

The phase of the complex filtered image can be used to analyze the structure that causes the rise to a given orientation. If the phase is 0, the energy corresponds to a step edge, while a positive or negative phase corresponds to a bright or dark line (bright line on dark background or dark line on bright background). Higher order

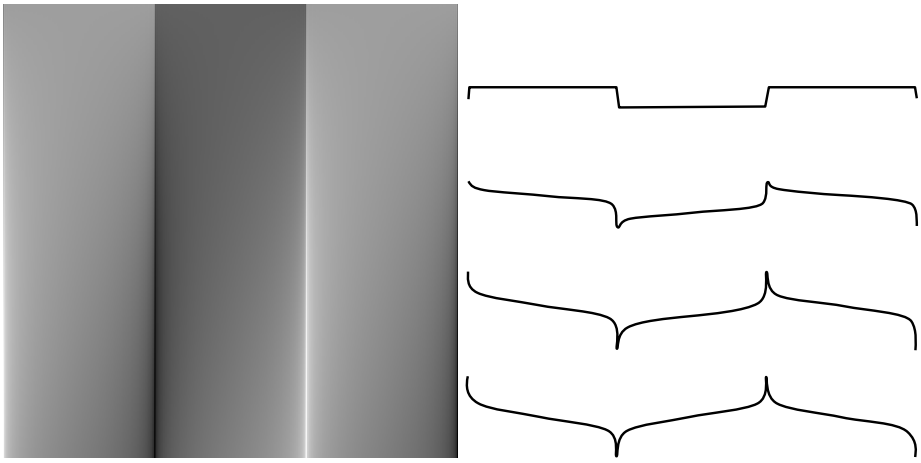


FIGURE 4.5: On the left is an image with a transition between line and step edge. On the right, the 1D intensity functions corresponding to the image are plotted.

FIGURE 4.6: Edge response of the Gaussian first derivative operator on the left; local energy response of the steerable Gaussian quadrature operator on the right. Both using a 5x5 filter kernel.

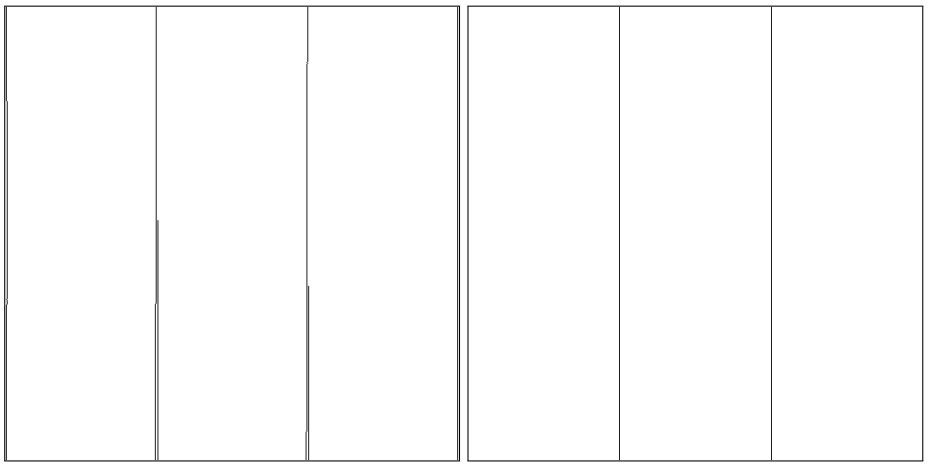


FIGURE 4.7: Non-maxima suppression of the Sobel operator on the left; non-maxima suppression of the local energy on the right.

quadrature filters (e.g., fourth derivative Gaussian of Gabor with smaller wavelength) can detect multiple simultaneous orientations in images, such as those occurring at intersecting lines or corners [62].

4.2.2 Combining Multiple Directions

To obtain a final local energy measure of a 2D signal, rotationally symmetric filters such as the Laplacian are required. Since the odd-symmetric filters can't be created as rotationally symmetric filters (see subsection 4.2.5 for more details), multiple orientations of the quadrature pairs have to be applied to an image. This creates two problems that need to be solved: How many orientations to use and how to combine the responses.

For the first problem, a filter bank should uniformly tile the frequency plane and achieve uniform coverage of the 2D spectrum. A Gaussian filter envelope should be used for a smooth transition. Kovesi suggests using 6 directions for a log Gabor filter bank [113]. More about filter orientation will be discussed in the next section.

For the second problem, there are several possible solutions. A simple solution is to sum the filter results of all directions:

$$\begin{gathered} e(x,y) = \sum_i e^{\theta_i}(x,y), \quad o(x,y) = \sum_i o^{\theta_i}(x,y) \\ E(x,y) = \sqrt{e(x,y)^2 + o(x,y)^2} \\ \varphi(x,y) = atan2(o(x,y), e(x,y)) \end{gathered} \tag{4.26}$$

An estimate of the strongest direction of the feature can be calculated, by summing up the decomposed x and y part of all odd filter responses:

$$\begin{gathered} o_x(x,y) = \sum_i \cos(\theta_i) o^{\theta_i}(x,y), \quad o_y(x,y) = \sum_i \sin(\theta_i) o^{\theta_i}(x,y) \\ \theta_s(x,y) = atan2(o_y(x,y), o_x(x,y)) \end{gathered} \tag{4.27}$$

A more general approach to analyze how local energy varies with orientation is to apply Principal Component Analysis (PCA) [100]. This can be done by constructing the covariance matrix of image gradients, which in this context is the second moment matrix (also known as the structure tensor). It describes the local structure of an image by:

$$M = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix}, \tag{4.28}$$

where I_x and I_y are the image intensity derivatives in x- and y-direction, summed over a local neighborhood (optionally with a weighting function). The elements of M capture the second-order moments of local gradient variations. Since M is a symmetric 2×2 matrix, it can be diagonalized directly [74, 92].

The eigenvectors of M correspond to the principal axes of the local gradient distribution and indicate the dominant orientations of intensity change [114]. The eigenvalues $\lambda_{\max}$ and $\lambda_{\min}$ (principal moments) quantify the magnitude of variation along these axes:

- $\lambda_{\max}$: strongest local variation (feature importance),
- $\lambda_{\min}$: weakest local variation; if large, indicates a strong 2D structure such as a corner or junction.

An equivalent formulation can be derived when the local gradients are expressed in terms of their magnitude E^{θ_i} and direction θ_i over N samples in the neighborhood. In this case, the elements of M can be written as:

$$\begin{aligned} a &= \sum_{i=1}^{N} \left(E^{\theta_i} \cos\theta_i\right)^2, \\ b &= 2\sum_{i=1}^{N} \left(E^{\theta_i} \cos\theta_i\right)\left(E^{\theta_i} \sin\theta_i\right), \\ c &= \sum_{i=1}^{N} \left(E^{\theta_i} \sin\theta_i\right)^2. \end{aligned} \tag{4.29}$$

Here, $a = \sum I_x^2$, $b = 2\sum I_x I_y$, and $c = \sum I_y^2$ correspond exactly to the entries of the second moment matrix:

$$M = \begin{bmatrix} a & b/2 \\ b/2 & c \end{bmatrix}.$$

From these coefficients, the principal axis orientation is given by:

$$\theta_p = \frac{1}{2}\operatorname{atan2}(b, a-c), \tag{4.30}$$

and the maximum and minimum moments (identical to the eigenvalues of M) are:

$$\begin{aligned} M_{\max} &= \frac{a+c}{2} + \frac{\sqrt{b^2+(a-c)^2}}{2}, \\ M_{\min} &= \frac{a+c}{2} - \frac{\sqrt{b^2+(a-c)^2}}{2}. \end{aligned} \tag{4.31}$$

Both formulations therefore describe the same underlying quantities: the second moment matrix, its eigenvalues, and eigenvectors, which capture the dominant orientation and the strength of local image structures.

The resulting maps $\theta_p(x,y)$, $M_{\max}(x,y)$ and $M_{\min}(x,y)$ can be used for further processing to detect edges and corners.

Another approach, as discussed in the next section, is to approximate the direction with the strongest energy response and to use that local energy.

4.2.3 Steerable Filters

To obtain the response of a filter at many orientations, the same filter can be applied at different rotations. A more efficient way is to compute the orientation of a few filters at different rotations, and then use them to interpolate between arbitrary orientations. To do this, it is important to know how many rotated variations of the filter are needed and how to use them to interpolate between the responses. This results in a base filter set with a corresponding interpolation rule. Freeman et al.

[62] use the term steerable filters to describe a class of filters that can compute a filter of any orientation as a linear combination of a set of basis filters. They defined a steering constraint that describes the rule how to use the linear combination for interpolation:

$$f^{\theta}(x,y) = \sum_{j=1}^{M} k_j(\theta) f_b^{\theta_j}(x,y) \tag{4.32}$$

Three theorems are defined that describe how to find base functions $f_b(x,y)$ that satisfy the constraint, how many base functions M are needed in total for the linear combination, and what the interpolation functions $k_j(\theta)$ are.

The theorems show that all functions that can be expressed as a Fourier series (depending on θ) or as a polynomial expansion times a radially symmetric window function are steerable. An example is derivatives of Gaussians of all orders. For the first derivative, a set of basis filters is defined as follows

$$\begin{aligned} f^0(x,y) &= \nabla_x G(x,y) = \frac{\partial}{\partial x} c e^{-(x^2+y^2)} = -2cxe^{-(x^2+y^2)} \\ f^{\frac{\pi}{2}}(x,y) &= \nabla_y G(x,y) = \frac{\partial}{\partial y} c e^{-x^2+y^2} = -2cye^{-(x^2+y^2)} \end{aligned} \tag{4.33}$$

And compute a filter of any rotation with the rule:

$$f^{\theta}(x,y) = \cos(\theta)\, f^0(x,y) + \sin(\theta) f^{\frac{\pi}{2}}(x,y) \tag{4.34}$$

The terms $\cos(\theta)$ and $\sin(\theta)$ correspond to interpolation functions. Since the filters are applied by convolution, and convolution is a linear operation, it is also possible to apply the base filters to an image first, and then compute the interpolation:

$$\begin{aligned} F^0 &= f^0 * I \\ F^{\frac{\pi}{2}} &= f^{\frac{\pi}{2}} * I \\ F^{\theta} &= \cos(\theta)\, F^0 + \sin(\theta) F^{\frac{\pi}{2}} \end{aligned} \tag{4.35}$$

For steerable quadrature filters, a steerable filter basis set must be defined and the corresponding Hilbert transforms. The second derivative of the Gaussian is defined as:

$$f^0(x,y) = \Delta_x G(x,y) = c\left(4x^2 - 2\right) e^{-(x^2+y^2)} \tag{4.36}$$

Obviously, it is the product of a second order polynomial and a radially symmetric Gaussian window. So, applying theorem 3 of Freeman et al., three basis functions should be sufficient. A basis set could be found with: $\theta_0 = 0$, $\theta_1 = \frac{\pi}{2}$, $\theta_2 = \frac{2\pi}{3}$:

$$\begin{aligned} f^{\theta} &= k_0(\theta) f^0 + k_1(\theta) f^{\frac{\pi}{3}} + k_2(\theta) f^{\frac{2\pi}{3}}, \\ k_j(\theta) &= \frac{1}{3} + \frac{2}{3}\cos(2\theta - 2\theta_j) \end{aligned} \tag{4.37}$$

The corresponding Hilbert transforms can be found, e.g., by approximating the bases with a least-squares fit to a polynomial times a Gaussian. Freeman et al. found a third-order polynomial that approximates the Hilbert bases and is steerable. A third order polynomial requires four interpolation functions to interpolate between all the

rotations. Let H be the Hilbert transform of f in the direction of the derivative:

$$\begin{aligned} h^{\theta} &= l_0(\theta)H^0 + l_1(\theta)H^{\frac{\pi}{3}} + l_2(\theta)H^{\frac{\pi}{2}} + l_3(\theta)H^{\frac{2\pi}{3}}, \\ l_j(\theta) &= \frac{1}{2}\cos\left(\theta - \theta_j\right) + \frac{1}{2}\cos(3\theta - 3\theta_j) \end{aligned} \tag{4.38}$$

The bases of this quadrature steerable set are not separable. For performance reasons, a separable quadratic basis set would be preferable. A separable basis can be found by looking at the polynomial functions $f(x,y)$:

$$f^{\theta}(x,y) = \sum_i \sum_j k_{ij}(\theta)x^i y^j \tag{4.39}$$

This can lead to many basis functions, but in most cases there is a separable basis set that contains only the minimum number of basis filters. The second derivative of Gaussian is then defined as

$$\begin{aligned} f^0(x,y) &= \Delta_x G(x,y) = c\left(4x^2 - 2\right)e^{-(x^2+y^2)}, \\ f^{\pi}(x,y) &= \Delta_y G(x,y) = c\left(4y^2 - 2\right)e^{-(x^2+y^2)}, \\ f^{\frac{\pi}{2}}(x,y) &= 4cxye^{-(x^2+y^2)}, \\ f^{\theta} &= k_0(\theta)f^0 + k_1(\theta)f^{\frac{\pi}{2}} + k_2(\theta)f^{\pi}, \\ k_0(\theta) = \cos^2(\theta)\,,\ k_1(\theta) &= -2cos(\theta)sin(\theta)\,,\ k_2(\theta) = \sin^2(\theta)\,, \\ h^{\theta} &= l_0(\theta)H^0 + l_1(\theta)H_x^{\frac{\pi}{2}} + l_2(\theta)H_y^{\frac{\pi}{2}} + l_3(\theta)H^{\pi}, \\ l_0(\theta) = \cos^3(\theta)\,,\ \ l_1(\theta) &= -3cos^2(\theta)sin(\theta)\,, \\ l_2(\theta) = 3cos(\theta)sin^2(\theta)\,, l_3(\theta) &= \sin^3(\theta) \end{aligned} \tag{4.40}$$

$H_x^{\frac{\pi}{2}}$ and $H_y^{\frac{\pi}{2}}$ denote the Hilbert transform in the x and y directions, since no principal direction of the derivative is given for $f^{\frac{\pi}{2}}$.

Another approach to designing steerable filters is to work in the frequency domain by combining the radial and angular parts of the filter. An inverse Fourier transform can be used to determine the filter kernel, or operations can be performed in the frequency domain. For more information, see [115, 62] or the website of Kovesi[1].

The strength along a particular direction θ is calculated from the squared sums of the filter outputs, steered at the angle θ. It is called oriented energy:

$$E(\theta)^2 = \left(f^{\theta} * I\right)^2 + \left(h^{\theta} * I\right)^2 \tag{4.41}$$

This simplifies the Fourier series depending on the angle θ, where odd frequencies are removed by the squaring operation:

$$E(\theta)^2 = C_1 + C_2\cos(2\theta) + C_3\sin(2\theta) + \ldots \tag{4.42}$$

[1] http://www.peterkovesi.com/matlabfns/PhaseCongruency/Docs/convexpl.html

Then the dominant direction θ_d and its strength S (the orientation that maximizes E) can be approximated:

$$\theta_d = \frac{1}{2}\text{atan2}(C_2, C_3)$$
$$S = \sqrt{C_2^2 + C_3^2} \tag{4.43}$$

With θ_d the maximum local energy $E(\theta_d)$ can be calculated. The resulting gradient-like description of the image can be used for further processing (as edge regions) as described in the next chapter.

4.2.4 Phase Congruency (PC)

A major problem with the already discussed edge region methods is the dependency on illumination and contrast of images. Most approaches are also sensitive to image magnification and blur. Thresholds are needed to decide whether a feature is significant or not. Efforts to automatically determine the threshold are very limited (see [31, 116] and section 6.1). Typically, thresholds are determined empirically. To overcome the threshold problem, an invariant measure of the significance of features would be very useful.

Quadrature filters have already been investigated for feature detectors in the frequency domain. The maxima of the local energy E were used to determine the edge regions. Morrone et al. [153] stated that the points of maximum phase congruence (PC) correspond to the local maxima in E, since E is proportional to PC. But PC doesn't depend on local contrast, unlike E, which requires a threshold to find significant features.

Morrone et al. defined phase congruency in terms of the Fourier series expansion at location x as:

$$PC(x) = max_{\overline{\varphi}(x)} \frac{\sum_n a_n \cos(\varphi_n(x) - \overline{\varphi}(x))}{\sum_n a_n} \tag{4.44}$$

Where $\varphi_n(x)$ is the local phase at position x of the nth Fourier component and a_n is the amplitude of the nth Fourier component. PC is then defined by finding the amplitude-weighted mean local phase angle $\overline{\varphi}(x)$ of all Fourier terms at position x that maximizes the equation (peaks). Venkatesh et al. [221] showed that points of maximum PC can be determined from peaks in the local energy function. They are related as:

$$E(x) = PC(x) \sum_n a_n \tag{4.45}$$

Additionally, E is defined by the convolution of the signal with the quadrature filter pair $F(x)$ and $H(x)$:

$$E(x) = \sqrt{F(x)^2 + H(x)^2} \tag{4.46}$$

In Figure 4.8 the relationship between PC, E and a_n are illustrated. It can be seen that E is equal to the term of PC without the normalization:

$$E(x) = \sum_n a_n \cos(\varphi_n(x) - \overline{\varphi}(x)) \tag{4.47}$$

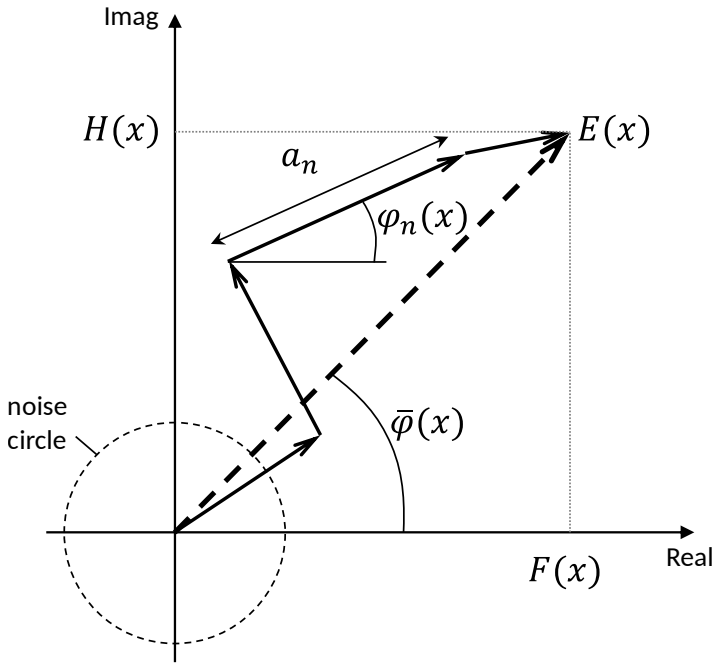


FIGURE 4.8: Illustration of the phase congruency components: The final responses $F(x)$ and $H(x)$ are the sum of the even and odd filter responses $e_n(x)$ and $o_n(x)$ of different scales, while the amplitude $a_n(x)$ and phase $\varphi_n(x)$ are the polar coordinates of the even and odd filter responses of scale n.

PC can be interpreted as the ratio of E to the total path length taken by the local Fourier components $\varphi_n(x)$ and a_n to reach the endpoint:

$$PC(x) = \frac{E(x)}{\sum_n a_n} \tag{4.48}$$

Kovesi [115] connected the significance of local energy maxima with phase congruency and saw that PC is independent of the overall magnitude of the signal. This provides invariance to variations in illumination and contrast in images.
To compute the phase congruency, Kovesi used quadrature filter pairs in multiple scales. As a convenient filter model, Field [58] proposed to use logarithmic Gabor functions, since they allow large bandwidth filters with zero DC component in the even-symmetric filter (see section A.1 for more details about DC component). If I is defined as the signal, and K_n^e and K_n^o are defined as the even and odd log Gabor filter kernels with scale n, a response vector can be formed:

$$[e_n(x), o_n(x)] = [I(x) * K_n^e, I(x) * K_n^o] \tag{4.49}$$

The amplitude and phase of the scale are n:

$$\begin{aligned} a_n(x) &= \sqrt{e_n(x)^2 + o_n(x)^2}, \\ \varphi_n(x) &= atan2(o_n(x), e_n(x)) \end{aligned} \tag{4.50}$$

An estimate of F and H can be calculated by summing the even and odd filter responses:

$$F(x) = \sum_n e_n(x), \quad H(x) = \sum_n o_n(x)$$
$$A(x) = \sum_n a_n(x) = \sum_n \sqrt{e_n(x)^2 + o_n(x)^2} \tag{4.51}$$

Kovesi addressed the following problems with phase congruency for feature detection:

1. If only very small frequency components are present in the signal, the calculation of phase congruency is ill-conditioned
2. Phase congruency is sensitive to noise because it is a normalized quantity
3. If only one frequency component is present, phase congruency degenerates to 1
4. The phase congruency measure provides a worse localization for features

His final phase congruency equation takes into account the enumerated problems:

$$PC_2(x) = \frac{\sum_n W(x) \lfloor a_n(x) \Delta\Phi_{n(x)} - T \rfloor}{A(x) + \varepsilon} \tag{4.52}$$

The ε in the denominator will prevent the equation from becoming ill-conditioned as stated in issue 1. The variable T represents the noise radius as shown in Figure 4.8 to reduce the sensitivity to noise as stated in issue 2. Kovesi uses the Rayleigh distribution to describe the noise energy response and to define the noise radius T. For issue 3, a weighting function $W(x)$ has been incorporated into the equation that corresponds to the frequency spread. Although reducing the noise responses and weighting the frequency spread improves the localization of the feature, it is still poor. To sharpen the phase deviation measure, the sine of the phase difference is incorporated:

$$\Delta\Phi_{n(x)=} \cos(\varphi_n(x) - \overline{\varphi}(x)) \; - |\sin(\varphi_n(x) - \overline{\varphi}(x))| \tag{4.53}$$

The brackets $\lfloor \; \rfloor$ in the equation mean that negative values are set to zero. For a more detailed explanation of the terms in the equation, see [115].

To compute the final phase congruency responses (energy, direction, and phase) for an image plane, the approaches described in subsection 4.2.2 can be applied. Kovesi proposes in [114] to use the second approach by determining the principal axis, since it can be used to additionally find corners. Later he suggested using monogenic filters to improve performance.

4.2.5 Monogenic Signal

A two-dimensional generalization of the analytical signal was introduced by Felsberg et al. [54]. The monogenic signal is a 2D isotropic analytic signal that uses the Riesz transform instead of the Hilbert transform. The Riesz transform can be viewed as a multidimensional generalization of the Hilbert transform. It is used to construct an odd or antisymmetric isotropic analytic signal that is vector-valued (multidimensional). As discussed earlier, a scalar-valued isotropic analytic signal can

only be expressed by a symmetric function (real or complex).

While the 1D analytic signal decomposes a signal into structural (local phase) and energetic (local amplitude) information, the monogenic signal decomposes the same information from a 2D signal and additionally the local orientation.

Similar to the 1D quadrature filter pair, a spherical quadrature filter (SQF) triplet can be designed to represent the monogenic signal. Thus, it is possible to extract all information with three convolutions in the spatial domain (or two complex multiplications in the frequency domain). In his thesis [55], Felsberg defined the spherical quadrature filter triplet as a radial symmetric kernel created by difference of Poisson (DOP) kernels at different scales and two odd symmetric kernels created by two difference of conjugate Poisson (DOCP) kernels at different scales and oriented along the X and Y axes:

$$\begin{aligned} b_{\lambda,k}(x,y;s) &= \frac{s\lambda^k}{2\pi\left(s^2\lambda^{2k}+x^2+y^2\right)^{\frac{3}{2}}} - \frac{s\lambda^{k-1}}{2\pi\left(s^2\lambda^{2k-2}+x^2+y^2\right)^{\frac{3}{2}}} \\ \boldsymbol{c}_{\lambda,k}(x,y;s) &= \left(\frac{1}{2\pi\left(s^2\lambda^{2k}+x^2+y^2\right)^{\frac{3}{2}}} - \frac{1}{2\pi\left(s^2\lambda^{2k-2}+x^2+y^2\right)^{\frac{3}{2}}}\right)\begin{pmatrix} x \\ y \end{pmatrix} \end{aligned} \tag{4.54}$$

The variable $\lambda \in [0,1]$ indicates the relative bandwidth, s the coarsest scale and $k \in \mathbb{N}$ the selected bandpass. If $\lambda = 0.5$, then k represents the octave. For simplicity, the convolution of an image with the even symmetric DOP kernel is called $e = I * b$ and the odd symmetric DOP kernels $o_x = I * b_x$ and $o_y = I * b_y$ or as a vector $\boldsymbol{o}$. Then, the orientation of at maximal local energy is:

$$\theta = atan2(o_y, o_x) \tag{4.55}$$

The phase can be computed for both orientations of the odd filters:

$$\varphi_x = atan2\left(o_x, e\right), \quad \varphi_y = atan2\left(o_y, e\right), \tag{4.56}$$

And the combined amplitude of the odd filters (gradient like edge strength) is:

$$E_\varphi = \sqrt{o_x^2 + o_y^2} \tag{4.57}$$

The phase φ at orientation θ is then defined as:

$$\varphi_\theta = atan2\left(E_\varphi, e\right) = \cos(\theta)\,\varphi_x + \sin(\theta)\varphi_y \tag{4.58}$$

The final local energy is defined as:

$$E = \sqrt{e^2 + o_x^2 + o_y^2} = \sqrt{e^2 + E_\varphi^2} \tag{4.59}$$

Felsberg proposed to use the scale derivatives of quadrature filters for edge detection [53]. This approach can also distinguish between points of phase congruency and others, similar to the approach of Kovesi's (see previous section). The resulting map contains zero crossings, indicating that there is no change of phase for a small change

of the scale. He defined the phase as a vector:

$$\boldsymbol{\varphi}(x,y;s) = \begin{bmatrix} \varphi_x(x,y;s) \\ \varphi_y(x,y;s) \end{bmatrix} \tag{4.60}$$

The phase derivatives of the scale must be zero for a stable phase:

$$\partial_s \boldsymbol{\varphi}(x,y;s) = 0 \tag{4.61}$$

The scale derivative can be expressed as:

$$\partial_s \boldsymbol{\varphi}(x,y;s) = \frac{e(x,y;s)\,\partial_s \boldsymbol{o}(x,y;s) - \boldsymbol{o}(x,y;s)\partial_s e(x,y;s)}{e(x,y;s)^2 + |\boldsymbol{o}(x,y;s)|^2} \tag{4.62}$$

With:

$$\partial_s \boldsymbol{o}(x,y;s) = I * \partial_s \boldsymbol{c}, \quad \partial_s e(x,y;s) = I * \partial_s b \tag{4.63}$$

The filter derivatives are:

$$\begin{aligned} \partial_s b_{\lambda,k}(x,y;s) = {} & \frac{\lambda^k}{2\pi\left(s^2\lambda^{2k} + x^2 + y^2\right)^{\frac{3}{2}}} - \frac{\lambda^{k-1}}{2\pi\left(s^2\lambda^{2k-2} + x^2 + y^2\right)^{\frac{3}{2}}} - \\ & \frac{3s^2\lambda^{3k}}{2\pi\left(s^2\lambda^{2k} + x^2 + y^2\right)^{\frac{5}{2}}} + \frac{3s^2\lambda^{k-1}\lambda^{2k-2}}{2\pi\left(s^2\lambda^{2k-2} + x^2 + y^2\right)^{\frac{5}{2}}} \end{aligned} \tag{4.64}$$

$$\partial_s \boldsymbol{c}_{\lambda,k}(x,y;s) = \left(\frac{3s\lambda^{2k-2}}{2\pi\left(s^2\lambda^{2k-2} + x^2 + y^2\right)^{\frac{5}{2}}} - \frac{3s\lambda^{2k}}{2\pi\left(s^2\lambda^{2k} + x^2 + y^2\right)^{\frac{5}{2}}} \right) \begin{pmatrix} x \\ y \end{pmatrix}$$

This method was also used by López et al. [133] for edge extraction within a stereo line tracking framework. For performance reasons, they skipped the normalization step, since zero crossings are independent of a scaling factor:

$$\partial_s \boldsymbol{\varphi}(x,y;s) = e(x,y;s)\,\partial_s \boldsymbol{o}(x,y;s) - \boldsymbol{o}(x,y;s)\partial_s e(x,y;s) \tag{4.65}$$

To produce nice zero crossings, the DC must be corrected to zero for e and $\partial_s e$ when computed as a convolution to avoid artifacts (see section A.1). As an alternative, the filter can be efficiently applied in the frequency domain without truncating the filter signal (see [55, pp. 97–100, pp. 151–157] for information on the required filter equations in the frequency domain). The final edges must be extracted separately for each response (for o_x and o_y), since they cannot simply be combined. Using a norm eliminates the zero crossings, and summing the signals can cause them to cancel each other out.

4.2.6 Conclusion

Local energy and phase congruency incorporate the perceptual importance of phase in images. They have the ability to detect different types of edges, such as step and roof edges. Phase congruency is also invariant to variations in illumination and contrast in images. However, they are more complex, require more computational time, and in most cases perform similarly to the simpler and faster differential approaches. The monogenic extension minimizes computational time by requiring

only three convolutions in the spatial domain or two complex multiplications in the frequency domain (and also only two inverse FFTs) to obtain local energy, local orientation, and local phase.

4.3 Statistical Methods

The previously discussed approaches are not robust to texture, in particular the differential methods are unable to detect boundaries defined by texture changes. Statistical methods can more easily overcome this problem by analyzing local patterns around image points.

One approach is to use a dissimilarity test that indicates the presence of a boundary region for a high value. A simple implementation is to divide the region around a pixel into equal parts along a given orientation and apply a two-sample test (difference of the means of the two regions) to measure the dissimilarity. This is done for multiple orientations, and the orientation with the maximum response is used as the boundary direction. Tests are: X^2, likelihood ratio, Wilcoxon, T-test, etc. [95, 36, 128]. This approach can be easily extended to color images by using color signatures [186].

Another approach is to analyze the gradient distribution around a given point. This can be done using a covariance matrix. The resulting eigenvalues and eigenvectors indicate contour strength and orientation [160, 6]. It is also possible to distinguish edges from lines and to simply analyze the gradient orientations within a region. The presence of a boundary is characterized by low angular dispersion [69]. These approaches can also be used for statistical analysis of directional texture if the area for analyzing the gradient orientation is much larger than the neighborhood used to compute the gradients.

A third approach is to use a similarity measure from a central pixel to a given area around the pixel. A high dissimilarity indicates a contour. A fairly fast and popular implementation of this approach is the SUSAN operator [198].

4.3.1 SUSAN Operator

The Smallest Univalue Segment Assimilating Nucleus (SUSAN) operator consists of the following steps:
First, within a circular mask, compute for each point the difference to the nucleus (center point) and test if the difference is significant:

$$\begin{aligned} n(x_0, y_0) &= \sum_{x_r} \sum_{y_r} c(x_r, y_r, x_0, y_0), \\ c(x_r, y_r, x_0, y_0) &= \begin{cases} 1 \; if \; \left| I(x_r, y_r) - I(x_0, y_0) \right| \leq t \\ 0 \qquad\qquad\qquad\qquad\qquad else \end{cases} \end{aligned} \tag{4.66}$$

Alternatively, a weighted window can be used:

$$c(x_r, y_r, x_0, y_0) = e^{-(\frac{I(x_r, y_r) - I(x_0, y_0)}{t})^2} \tag{4.67}$$

The parameter t determines the minimum significance a feature must have for detection and the maximum amount of noise that will be ignored.

The final edge response is then calculated as:

$$R(x_0, y_0) = \begin{cases} g - n(x_0, y_0) \ if \ n(x_0, y_0) < g \\ 0 \qquad\qquad\qquad\qquad\quad else \end{cases} \tag{4.68}$$

Where g is defined as $\frac{3}{4} n_{max}$.
If the edge response is greater than 0, the direction can be computed. Different cases have to be considered, but by default the orientation can be computed by the difference of the nucleus and the center of gravity, since the result vector has perpendicular direction to the edge:

$$G(x_0, y_0) = \left[\frac{\sum_{x_r} \sum_{y_r} x_r c(x_r, y_r, x_0, y_0)}{n(x_0, y_0)}, \frac{\sum_{x_r} \sum_{y_r} y_r c(x_r, y_r, x_0, y_0)}{n(x_0, y_0)} \right]^t \tag{4.69}$$

The resulting edge response and edge direction can be used for further processing such as non maxima suppression. The edge detector is robust to noise and has a good structure preserving property. With some small extensions, the SUSAN edge detector can also be used for corner detection.

4.3.2 Conclusion

The statistical approaches can be very effective for boundary detection in images containing noise or texture, but are computationally more demanding and not as well understood as the derivative methods.

4.4 Morphological Methods

Morphological operators were originally developed for binary images. The basic concept in binary morphology is to test an image with a predefined shape. The resulting information gives a conclusion about how the given shape fits to shapes in the image. This shape is called a structuring element and is also a binary image.

The two main operations are erosion and dilation [205, chapter 3.3.2]. Let E be a Euclidean space and f a binary image in E. Let b be the structuring element (e.g. a 3x3 mask with ones), then the two operations can be defined as follows:

1. Erosion:
$$f \ominus b = \bigcap_{x \in b} f_{-x} \tag{4.70}$$

2. Dilation:
$$f \oplus b = \bigcap_{x \in b} f_x \tag{4.71}$$

These two operations can be used to detect contours in binary images by first dilating the image and then subtracting an eroded image. This is called the morphological gradient [179]:

$$\nabla f = f \oplus b - f \ominus b \tag{4.72}$$

For grayscale images, the erosion and dilation operations are defined as follows:

$$\begin{aligned}(f \ominus b)(x) &= \inf[f(y) - b(y-x)], \quad y \in E \\ (f \oplus b)(x) &= \sup[f(y) + b(x-y)], \quad y \in E\end{aligned} \tag{4.73}$$

Where inf is the infimum (lowest value within the structuring element) and sup is the supremum (highest value within the structuring element). The function $f(x)$ is the intensity function of the image and $b(x)$ is the structuring function. Using a flat structuring element b (simple binary mask) simplifies both operations. For the discrete case, supremum and infimum can be replaced by maximum and minimum. The final equation for the morphological gradient, using a flat symmetric structuring function b, is then defined as follows:

$$\nabla f = \max[f(x)] - \min[f(y)] = \max\left(\left|f(x) - f(y)\right|\right), \ \forall x, y \in b \tag{4.74}$$

In this case, the structuring function b can be viewed as a sliding window over the image f that computes the maximum intensity difference of all points within the window centered on any point in the image. Compared to other methods, this approach can be easily extended to color images by replacing the absolute value in the equation with a norm in the color space [50].

The main drawbacks of the classical morphological gradient are its sensitivity to noise (it simply uses the maximum differences of two intensities within a window) and the lack of information about the edge direction.

4.4.1 RCMG Operator

The Robust Color Morphological Gradient (RCMG) operator [51] uses a special metric applied directly to the RGB color space to replace the absolute value in the CMG equation. The RCMG also addresses the problem of noise in images and the need for edge directions for most post-processing.

By simply using the maximum intensity difference within a sliding window as a gradient, this classic CMG approach is prone to noise. A simple method to increase robustness is to sort the differences and use the median instead of the maximum intensity difference [219]. Another effective approach is to compute the minimum vector dispersion [213]. However, considering the computational time and the possibility to efficiently compute the orientation of an edge, the mentioned methods are not suitable. Evans et al. proposed a pairwise pixel rejection scheme [51], by removing the pixels that are furthest apart (in terms of intensity). The rejection process can be repeated several times. This results in a gradient that is less affected by image noise. The number of useful rejected pairs depends on the window size of b. For a 5x5 window size, the authors recommend using a rejection size between 4 and 9.

To estimate the edge direction, the authors of RCMG proposed a method inspired by the LUV gradient of Shafarenko et al. [195]. They simply compute the line from the pixel position of the pair selected as the gradient by intensity difference. The normal of this line gives a rough estimate of the edge direction (error $< \frac{\pi}{4}$), which is sufficient for fast non-maximum suppression.

4.4.2 Conclusion

The morphological edge detectors are less computationally expensive than most statistical methods, but also incorporate some of their aspects by computing value ranks (sorting and weighting within the structuring window). They also have some similarities to differential methods, but can't provide good directional estimation. The easy way to incorporate an arbitrary norm into the basic equation makes them a good choice for edge processing of color images.

4.5 Even more Methods

There are even more methods for edge extraction. They are combinations of several existing local methods [149, 111, 148, 40] or more global methods (see subsection 4.5.1). In most cases they are used in a more complex framework to extract optimized bounds. Due to the higher computational complexity or the requirement of some user provided initial data, they are less suitable for real-time applications.

4.5.1 Global Methods

Global methods are discussed only in very general terms, since in many cases they are additional processing steps to the local edge region detection discussed in the next chapters, or they are not suitable for our purposes because of their high complexity.

Papari et al. [167] divided them into three main categories:

1. Contour saliency: it is motivated by psychological and neuropsychological studies [171, 150] which show that human visual perception of an orientation stimulus is influenced by other stimuli in the environment. Influences are the facilitation of stimuli that are collinear to a central stimulus and the inhabitation by other stimuli. This knowledge inspired the authors to develop post-processing methods that apply enhancements to the extracted local edge strength by analyzing the surround context. Surround suppression approaches have been developed that can enhance contours by reducing texture and noise [71, 70, 168]. Methods for contour facilitation have also been carried out, e.g. by using tensor voting [120, 80, 222]. While the two mentioned approaches are realized in a local context, they can be computed quite efficiently. More global approaches are contour integration using tensor voting [220, 157] or relaxation labeling, which is a probabilistic graph-based approach [180, 23, 169, 126].

2. Contour grouping: It is based on the grouping of pixels according to the Gestalt laws of proximity, good continuation, closure, and symmetry [48, 201, 228, 152, 224, 49]. The mathematical framework used is graph theory [19]. The nodes of a graph G correspond to contiguous edge pixels and the graph edges, which are connections between nodes, represent potential connections between edge pixel groups. A contour C can be described as a subgraph of G and is detected by minimizing a cost function over G. Contour grouping is usually built on top of the final localized edge pixels, as described in the next chapters. It can be used as post-processing on connected edge regions or simplified geometric objects such as line segments to detect gaps or patterns in edge region groups.

3. Active contours: In the literature this approach is also known as snakes or deformable models and was introduced by Kass et al. [103]. The basic concept

of this approach is to optimize a user-drawn curve around an object or along a contour by minimizing an energy function. The energy function typically consists of two terms, a smoothness energy, which measures the smoothness of the curve, and a fit energy, which measures how close the curve fits to the object or contour. They have the advantage that they can include multiple constraints such as continuity, smoothness, and closure, and that they are able to detect illusory contours as noted by Kass et al. This approach also has many problems such as dependence on initial conditions and slow convergence. Many additional approaches have been conducted based on the work of Kass et al. [202, 37, 79], addressing the problems.

4.5.2 Corner Regions

Corners and junctions can be used to reconstruct junctions that are lost by edge localization processes such as non-maximum suppression. For this reason, a brief introduction to corner and junction detection is given in this section.

The basic concept of corner or junction detection is to find points in the image with two or more different edge directions in the local region of the point. Based on this concept, Harris et al. [82] proposed a method that can estimate a corner response for each point in an image by using a second momentum matrix or structure tensor A. It uses partial derivatives of the image of a small subregion and relates them by the sum of squared differences (SSD). Similar to a covariance matrix (which is the inverse of A), the principal components can be estimated by computing the eigenvalue decomposition of the matrix A (see subsection 8.4.2 for more information). The eigenvalues represent the dispersion along the two principal axes. If both are large, there are multiple directions. A simple corner strength energy can be calculated by simply multiplying both eigenvalues.

With two additional steps, the SUSAN operator (also see subsection 4.3.1 for details) [198] can be adapted for corner detection:

1. **Reduction of the univalent segment size:** The minimum size of the univalent segment g is reduced to half the threshold used for edge detection.
2. **Centroid offset evaluation:** To mitigate false positives potentially introduced by step 1, the centroid of the USAN (Univalue Segment Assimilating Nucleus) is computed, and its distance to the nucleus is evaluated. True corners are characterized by a centroid noticeably offset from the nucleus, whereas false positives typically do not show such an offset.

4.6 Results and Discussion

Since this chapter is very extensive and presents many approaches to edge detection, a separate chapter has been dedicated to a comparison and evaluation of the methods.

Chapter 5

Edge Region Detector Comparison

To perform a comprehensive analysis of the introduced edge region detectors, a comprehensive framework has been implemented in C++ (see chapter 10 for more information). The framework is based on the OpenCV[1] library and uses (where possible) the optimized methods of OpenCV. The framework includes the following edge region detectors:

- Simple gradient-based filters: Roberts, Prewitt, Sobel, Scharr, and Gaussian first derivative (also includes Poisson filter via SQF)
- Simple Laplacian-based filters: Laplace, Laplacian of Gaussian (LoG), and Sobel-based Laplacian (via OpenCV)
- Local Energy Filters: Steerable quadrature filters (Freeman, QF StG X) and spherical quadrature filters (Poisson and log Gabor, SQF[F] X)
- Phase Matching Filters (PC): Kovesi and Felsberg methods
- Statistical Filters: SUSAN
- Morphological Filters: RMG and RCMG

For the phase congruency method, the original monogenic variant from Matlab was ported to C++ and integrated into the framework (PC ML). Additionally, an OpenCV-optimized variant has been implemented (PC). Both work in the frequency domain. For the Poisson-based spherical quadrature filter, two variants were implemented. One operates in the spatial domain (SQF PO), the other in the frequency domain (SQFF PO). The freely available sources of the SUSAN operator and the RCMG operator were ported to C++ and added to the framework. The RCMG was also optimized for grayscale images (RMG).

5.1 Runtime Comparison

To compare the runtime between the edge detectors, a set of images with different resolutions is used. For low resolution (481x321), the images from the Berkeley dataset [7] (BSDS500) are used. For higher resolutions: 750x500 (MDB-Q), 1500x1000 (MDB-H), and 3000x2000 (MDB-F), the images from the Middlebury stereo evaluation dataset (version 3)[2] [192] are used[3].

[1] http://opencv.org/

[2] http://vision.middlebury.edu/stereo/submit3/

[3] Left image only from stereo sets. ArtL, Computer and Teddy removed due to low resolution.

Method	BSDS500	MDB-Q	MDB-H	MDB-F
Roberts (2x2)	0.757	1.112	4.045	15.500
Prewitt (3x3)	0.560	0.824	2.950	11.530
Scharr (3x3)	0.437	0.764	2.944	11.510
Sobel (3x3)	0.354	0.664	2.631	10.461
Sobel (5x5)	1.111	2.202	8.686	34.574
Sobel (7x7)	1.680	3.366	13.357	52.521
Sobel (9x9)	2.391	4.707	18.547	71.819
Gauss (3x3)	0.861	1.786	7.371	28.706
Gauss (5x5)	1.278	2.582	10.241	40.923
Gauss (7x7)	1.753	3.433	13.923	55.302
Gauss (9x9)	2.137	4.285	17.339	69.031
QF StG (3x3)	12.220	35.117	141.703	574.034
QF StG (5x5)	13.524	37.830	152.428	614.626
QF StG (7x7)	15.103	40.985	165.224	665.408
QF StG (9x9)	16.555	43.926	177.610	713.872
SQF PO (3x3)	2.465	5.173	20.636	82.108
SQF PO (5x5)	6.615	13.719	54.606	217.830
SQF PO (7x7)	13.143	27.146	108.212	431.838
SQF PO (9x9)	8.966	14.774	54.327	208.536
SQFF LG	7.972	19.280	80.879	345.569
SQFF PO	7.638	19.167	80.854	343.584
PC	29.660	81.019	341.084	1,427.080
PC ML	323.536	1,257.681	5,540.054	27,748.375
Susan (37)	6.714	11.401	40.061	139.590
Susan (3x3)	2.276	3.775	12.724	40.611
RMG (3x3)	62.640	120.940	471.932	1,764.982
RMG (5x5)	527.752	1,025.122	4,006.871	15,195.238
RCMG (3x3)	105.740	219.280	866.868	3,350.292

Table 5.1: Execution time (ms) for gradient based edge detectors: Row "Roberts" to "Gauss" are first derivative based gradients, QF StG are quadrature based gradients (steerable filters), SQF PO are spherical quadrature filter gradients with Poisson kernels, SQFF LG and SQFF PO are spherical quadrature filters using log Gabor (LG) and Poisson (PO) kernels in the frequency domain, PC and PC ML are phase congruency gradients, Susan is a statistical gradient-like method, and RMG and RCMG are based on morphological gradients.

Method	BSDS500	MDB-Q	MDB-H	MDB-F
Laplace (3x3)	0.322	0.629	2.520	9.519
Laplace Sobel (5x5)	1.051	2.068	8.041	32.147
Laplace Sobel (7x7)	1.764	3.392	13.604	55.256
Laplace Sobel (9x9)	2.167	4.244	17.132	68.918
Laplace LoG (3x3)	1.016	1.984	7.804	32.512
Laplace LoG (5x5)	2.674	5.140	20.249	85.426
Laplace LoG (7x7)	4.981	9.818	39.205	164.406
Laplace LoG (9x9)	3.005	4.718	17.570	70.603
Laplace SQF (3x3)	5.523	12.540	50.799	210.655
Laplace SQF (5x5)	14.048	29.757	119.690	497.428
Laplace SQF (7x7)	27.508	56.722	227.376	941.047
Laplace SQF (9x9)	19.033	31.835	119.150	475.876
Laplace SQFF	13.366	33.749	142.612	628.626

Table 5.2: Execution time (ms) for Laplace-based edge detectors: The Laplace SQF and SQFF methods are based on phase derivatives as introduced in subsection 4.2.5. SQF is computed in the spatial domain, while SQFF is computed in the frequency domain.

5.1.1 Hardware Setup

All tests are performed on a mobile machine equipped with an Intel Core i5-3220M CPU (2.6 GHz) and 8 GB RAM, compiled on Windows 10 using Visual Studio 2015 and OpenCV 3. All methods are applied with optimized and feasible settings (see section A.1 for more details). For example, the simple methods are using a "short integer" 16 bit type to represent the gradient or Laplace data type. This enables the compiler to optimally use the SIMD instructions more efficient parallel processing. The Gaussian gradient uses float (32 bit) as data type to be more accurate than the Sobel or Scharr operator.

5.1.2 Analysis

The results in Table 5.1 and Table 5.2 show that most of the methods presented in this thesis can be performed in real time, at least for low resolution images. Figure 5.1 shows a table of the fast methods. The winner is the simple Laplace operator. It requires only a convolution and no additional processing to compute the magnitude, since the results of the Laplace can be used directly with zero crossing (see subsection 6.2.3). However, less information is provided, since no orientation information is given. In addition, the operator is not separable, so the runtime increases more steeply with increasing image size (compared to Sobel with only one convolution and without calculating the magnitude). The first derivative based methods are the fastest after the simple Laplace operator (squared magnitude computation is included), still leave enough time for extensive post-processing, and provide orientation data. Like the Gaussian operator, the SQF PO operator uses floating point precision for computation. But since it uses non-separable filter kernels, the runtime increases much faster for larger image sizes and filter kernels (see also Figure 5.18). The only exception to the fast non-derivative methods is the small SUSAN operator, which is also quite fast.

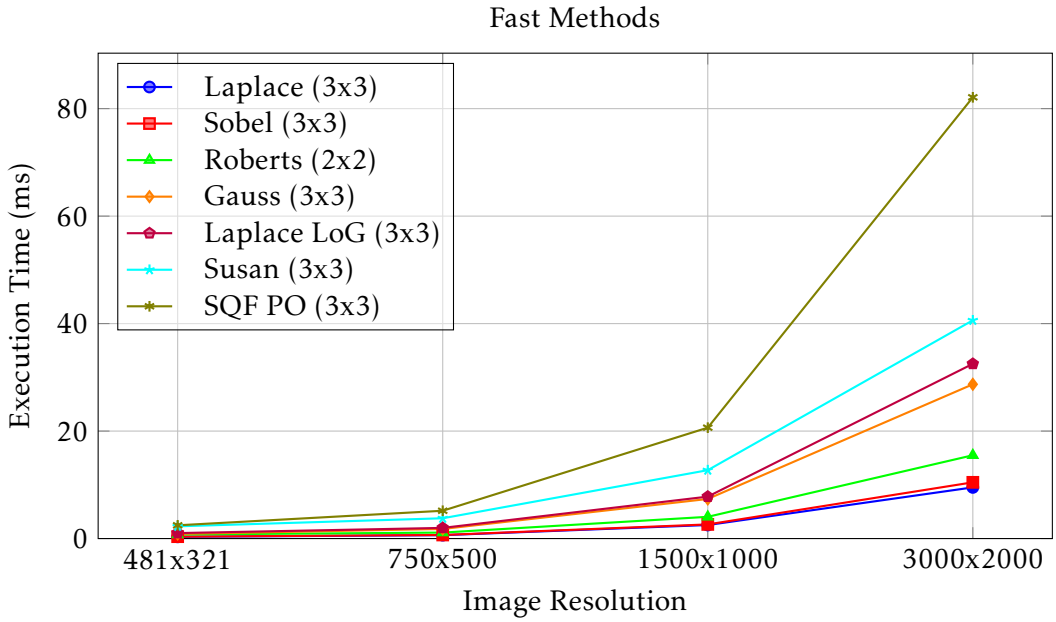


FIGURE 5.1: Chart of fast methods that can run in real time even for high-resolution images on the test system.

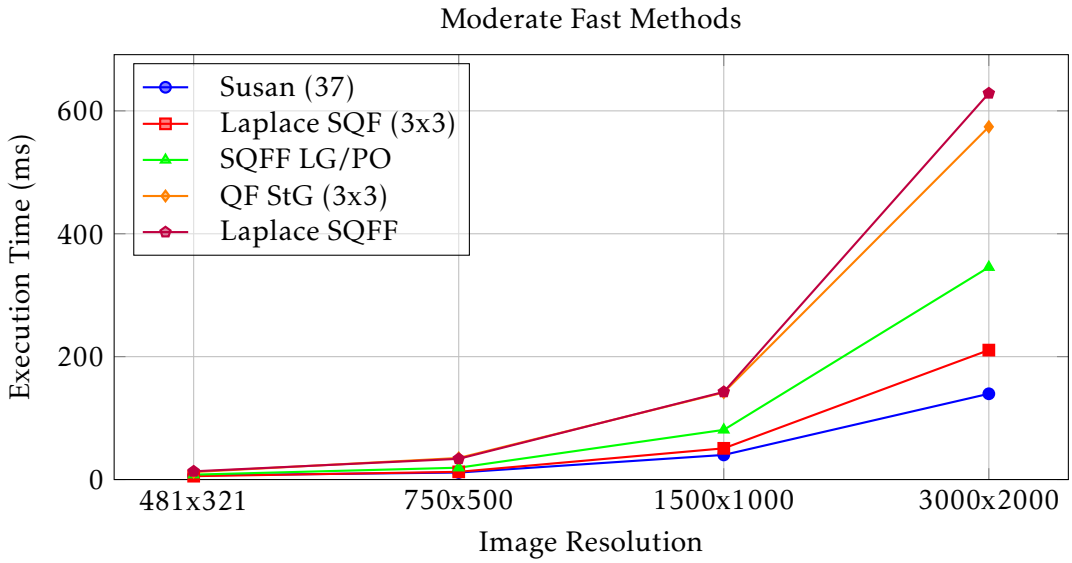


FIGURE 5.2: Chart of methods that can run in real time for low and medium resolution images on the test system.

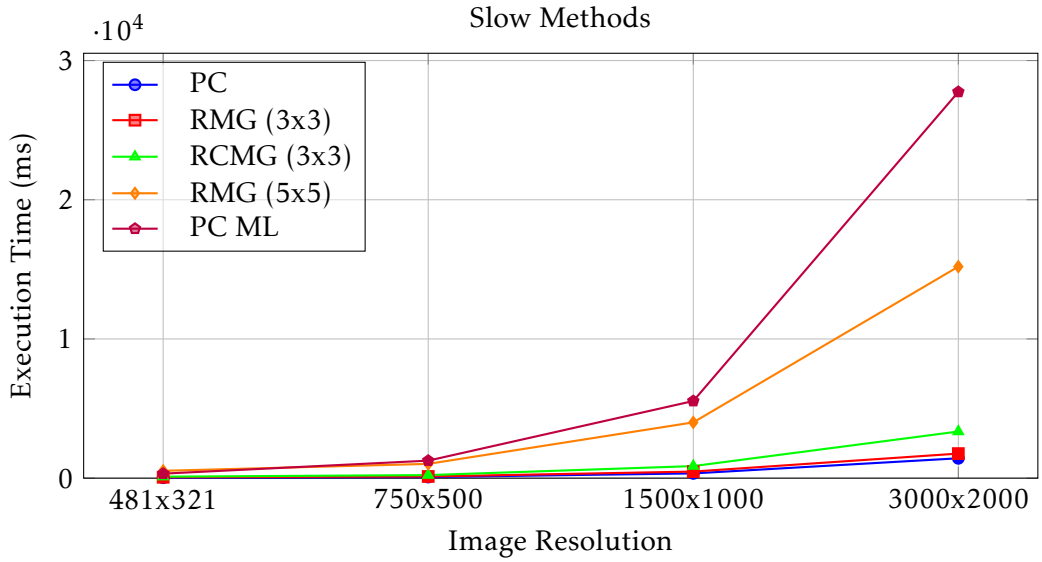


FIGURE 5.3: Chart of slow methods, some of which can run in real time on low-resolution images on the test system. The PC ML, RMG 5x5, and RCMG methods are not suitable for real-time use even at low resolution without further optimization or faster hardware.

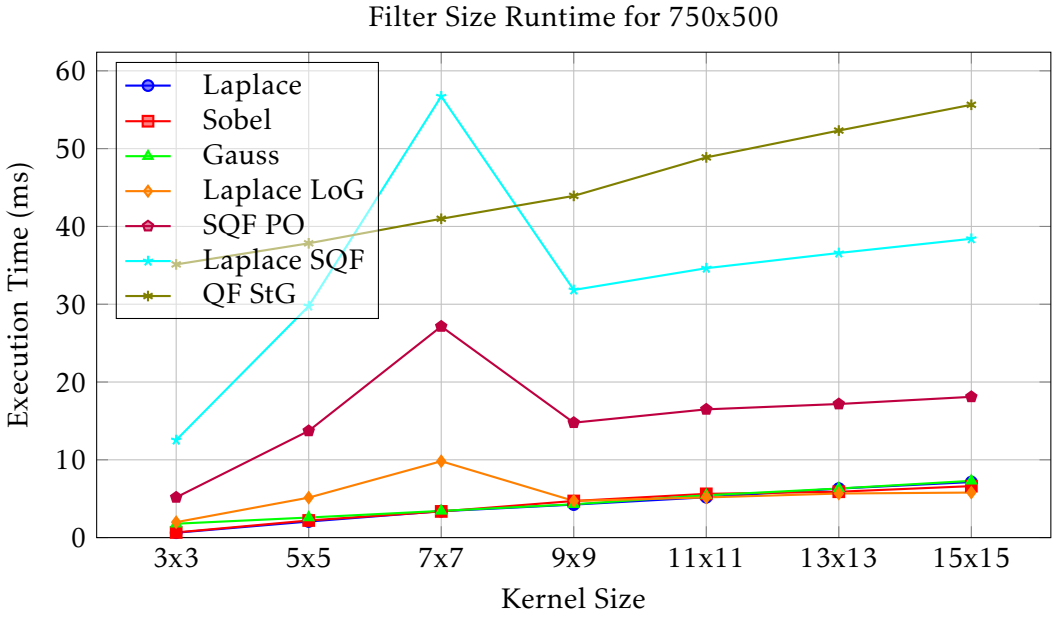


FIGURE 5.4: Chart of execution time (ms) for filters with increasing kernel size for low-resolution images.

Figure 5.1 shows a chart of moderately fast edge response methods. With the exception of the SUSAN operator, the other methods consist of multiple filtering operations in floating precision, which leads to a rapid increase in runtime for larger images, but also provides more information such as phase or local energy. The graph in Figure 5.3 shows even more complex methods like phase congruency, which also requires a scale space. The RMG and RCMG methods perform the worst, but are not highly optimized, similar to the PC ML method they are just exported from Matlab as C++ code.

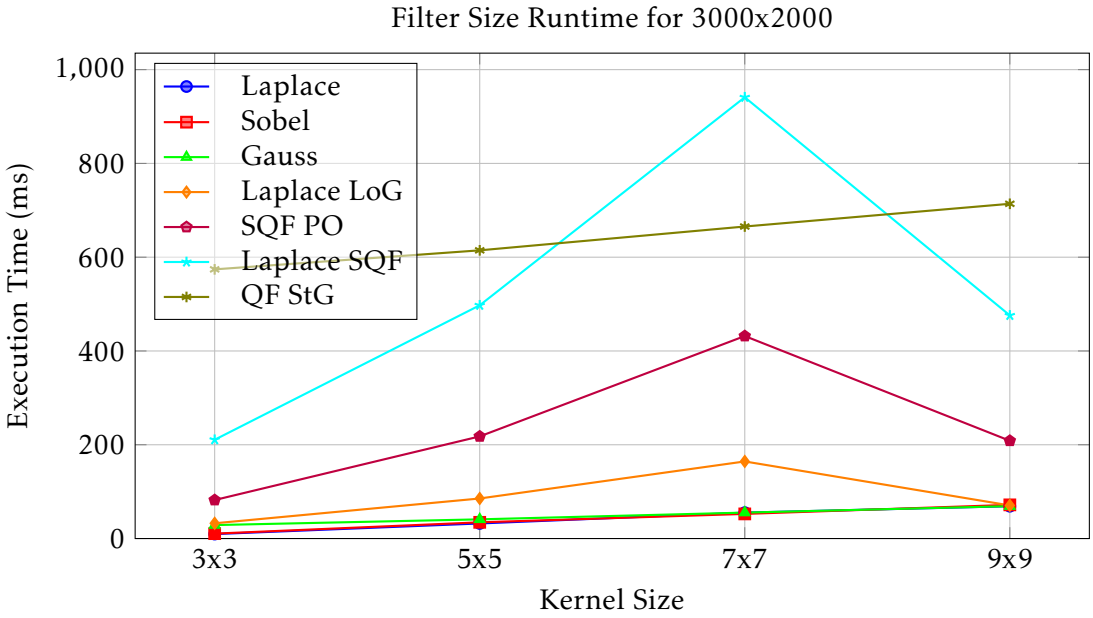


Figure 5.5: Chart of filter runtime (ms) with increasing kernel size for high-resolution images.

The graph in Figure 5.4 shows the runtime (in milliseconds) for the methods with different filter kernel sizes. The non-separable filters used for Laplace LoG, SQF PO, and Laplace SQF show a quadratic growth, while the other methods show a linear-like growth. For sufficiently large kernels (9x9), OpenCV switches from the spatial to the frequency domain for the filter computation to improve the runtime, since the fast Fourier transform only needs $O(nlogn)$ instead of $O(n^2)$. This explains the decrease in runtime from the 7x7 to the 9x9 filter kernels. However, OpenCV does not officially document the exact threshold at which it switches from spatial convolution to an FFT-based implementation, and the observed behavior is based on empirical measurements rather than guaranteed API behavior.

Figure 5.5 shows a smaller set of execution time for increasing filter kernel sizes on high-resolution images. The result shows a similar behavior as for the low-resolution images.

5.1.3 GPU Acceleration

GPU-based systems can benefit from the ability to perform massively parallel executions as long as the read and write operations can also be performed in parallel. This is also true for edge response methods, so they can benefit from this parallelism

since they are based on local operations. The OpenCV library provides a GPU module that can be used to accelerate edge response methods. However, the current implementation of the GPU engine does not support all methods and is not optimized for all hardware. For example, the GPU implementation of the Gaussian blur operator does not appear to be as efficient as the CPU implementation, making it attractive only for very large image sizes and/or large kernel sizes (see Figure 5.6 and Figure 5.7).

To analyze GPU behavior for OpenCV operations, two variants were selected that provide efficient CPU and GPU implementations (CUDA and OpenCL):

- **Gaussian Blur**: Used as an example of convolution operators.
- **Fast Fourier Transform**: Used as an example for frequency domain operations, which are also important for calculating edge response.

The tests were performed on a high performance HP ZBook 15 Fury G8 notebook with 32GB RAM, Intel Core I7 CPU (11th Gen i7-11850H @ 2.50GHz, 8 Cores) and NVIDIA RTX A3000 Laptop GPU, running ubuntu 22.04 (jammy). As in the previous experiments, the algorithms were executed based on the datasets BSDS500 and MDB-X.

Gaussian Blur

Method	BSDS500	MDB-Q	MDB-H	MDB-F
CPU 3x3	0.025	0.043	0.118	0.314
CPU 7x7	0.098	0.148	0.382	0.824
CPU 15x15	0.248	0.369	0.692	1.819
CPU 31x31	0.851	1.076	1.694	4.415
OCL 3x3	0.360	0.488	1.172	3.338
OCL 7x7	0.360	0.475	1.161	3.363
OCL 15x15	0.370	0.481	1.186	3.403
OCL 31x31	0.377	0.490	1.212	3.491
CUDA 3x3	0.156	0.189	0.499	1.305
CUDA 7x7	0.159	0.193	0.514	1.645
CUDA 15x15	0.167	0.226	0.579	1.983
CUDA 31x31	0.182	0.249	0.775	2.392
OCL NT 3x3	0.151	0.233	0.518	1.415
OCL NT 7x7	0.147	0.224	0.518	1.445
OCL NT 15x15	0.157	0.230	0.524	1.413
OCL NT 31x31	0.167	0.232	0.527	1.391
CUDA NT 3x3	0.019	0.025	0.075	0.254
CUDA NT 7x7	0.022	0.032	0.086	0.307
CUDA NT 15x15	0.030	0.047	0.131	0.530
CUDA NT 31x31	0.044	0.079	0.247	0.976

Table 5.3: Average execution time (ms) over data sets, based on the OpenCV Gaussian Blur operation. It includes three main variants: CPU, OCL (OpenCL) and CUDA based. The NT (no transfer) sub-variants (OCL NT and CUDA NT) show the timing without the transfer overhead of the image data to and from the GPU.

This experiment compares the execution time of the Gaussian blur operation on CPU and GPU. The results are shown in Table 5.3. Since the CPU implementation appears to be quite efficient, it is fast for small kernel sizes and moderate image sizes. The GPU implementation is faster for large image and kernel sizes. This is due to the overhead of transferring data between the CPU and GPU, which appears to be quite expensive for small images relative to the processing done on the GPU. However, for larger images, the parallel processing capabilities of the GPU can significantly reduce the execution time, which compensates for the transfer overhead at some point.

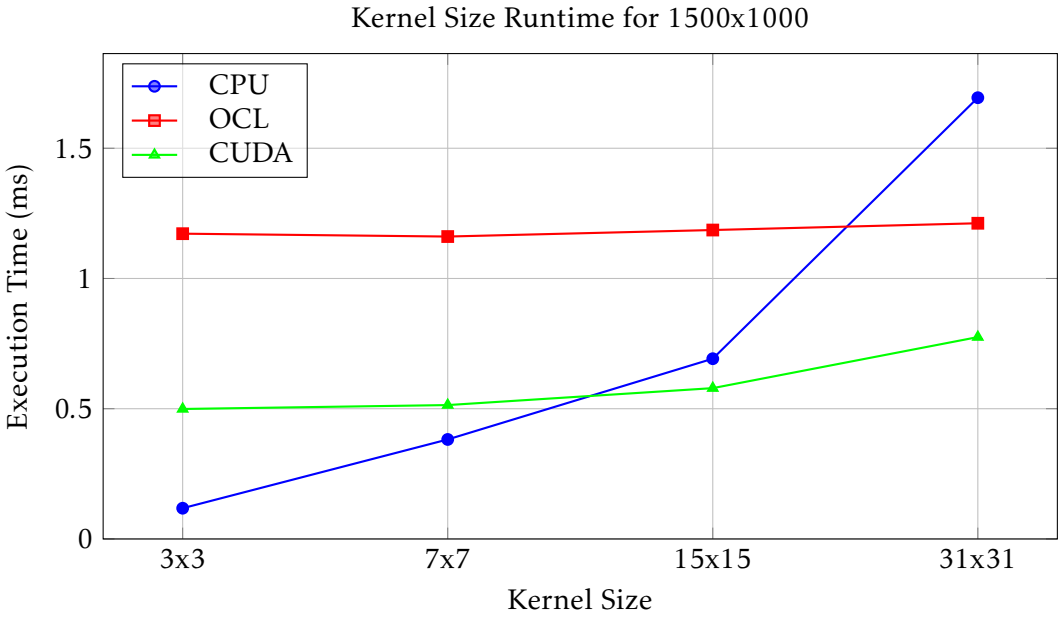


Figure 5.6: Chart of Gaussian blur runtime (ms) with increasing kernel size for half resolution images.

Figure 5.6 shows how the CPU runtimes for small kernel sizes start off well below the GPU runtimes. It takes a significant increase in kernel size for the GPU to outperform the CPU. The data transfer to the GPU seems to be much slower for the OpenCL variant. In this case, the kernel size has to be much larger to gain an advantage.

The Figure 5.7 also takes the image size into account. Interestingly, the efficiency of the CPU variant hardly decreases with increasing image size. In fact, it increases more slowly than the GPU variant because the transfer overhead for the GPU also increases with image size. Looking at the kernel size, the efficiency of the CPU implementation seems to suffer much more from it than the GPU implementations. They can more easily compensate for this by simply doing more parallel executions. In general, it looks like the CUDA implementation is more efficient than the OpenCL variant. But again, this seems to be due to the shorter transfer time. On the contrary, the increase in kernel size seems to have less impact on the OpenCL runtime than on the CUDA runtime.

If other convolution filters behave similarly, it does not make sense to switch to a GPU for general use. Unless the system has a very weak CPU but a strong GPU, or the use case involves working with large filter kernels or very large images, there is

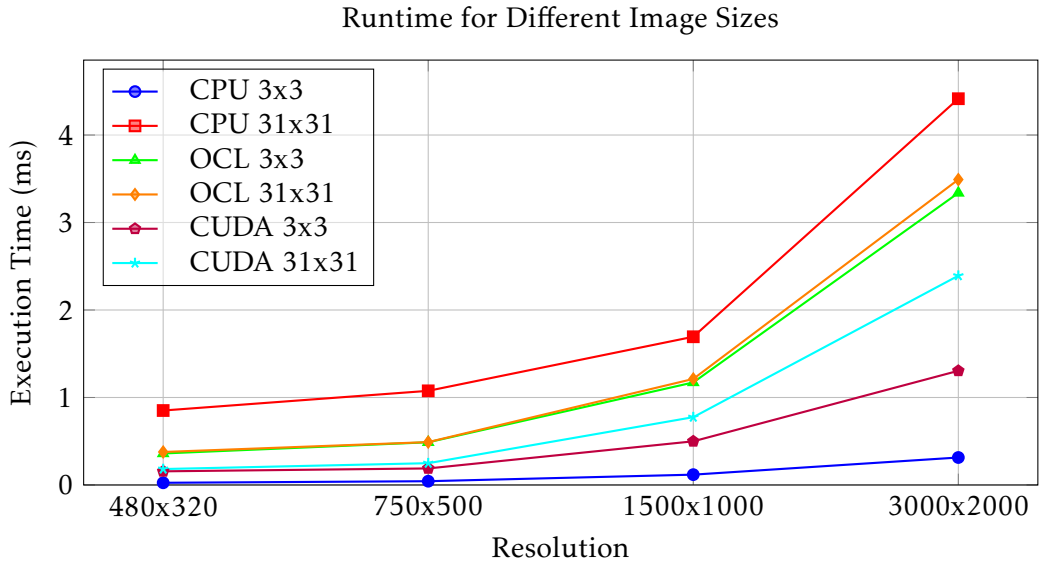


Figure 5.7: Chart of Gaussian Blur runtime (ms) with increasing image size.

no benefit in switching to a GPU-based convolution operator.

Fast Fourier Transform

The FFT is a widely used algorithm for computing the discrete Fourier transform (DFT) and its inverse. It is used in many applications, including image processing, where it can be used to filter images in the frequency domain. The OpenCV library provides an efficient implementation of the FFT algorithm that can be run on both CPU and GPU.

Method	BSDS500	MDB-Q	MDB-H	MDB-F
CPU	1.800	4.789	19.818	89.229
OCL	0.647	1.215	3.790	15.584
CUDA	0.579	0.895	2.847	12.974
OCL NT	0.149	0.324	0.872	3.267
CUDA NT	0.328	0.430	1.122	3.085

Table 5.4: Average execution time (ms) over data set, based on OpenCV FFT operation. There are three main variants: CPU, OCL (OpenCL), and CUDA based. The NT (no transfer) sub-variants (OCL NT and CUDA NT) show the timing without the transfer overhead of image data to and from the GPU.

The result observed in Table 5.4 is much more in line with expectations than in the previous example. The GPU variants are already 3 times faster than the CPU variant for small images, despite the transfer overhead. Figure 5.8 clearly shows how the CPU runtime increases much more steeply with increasing image size than the GPU variants.

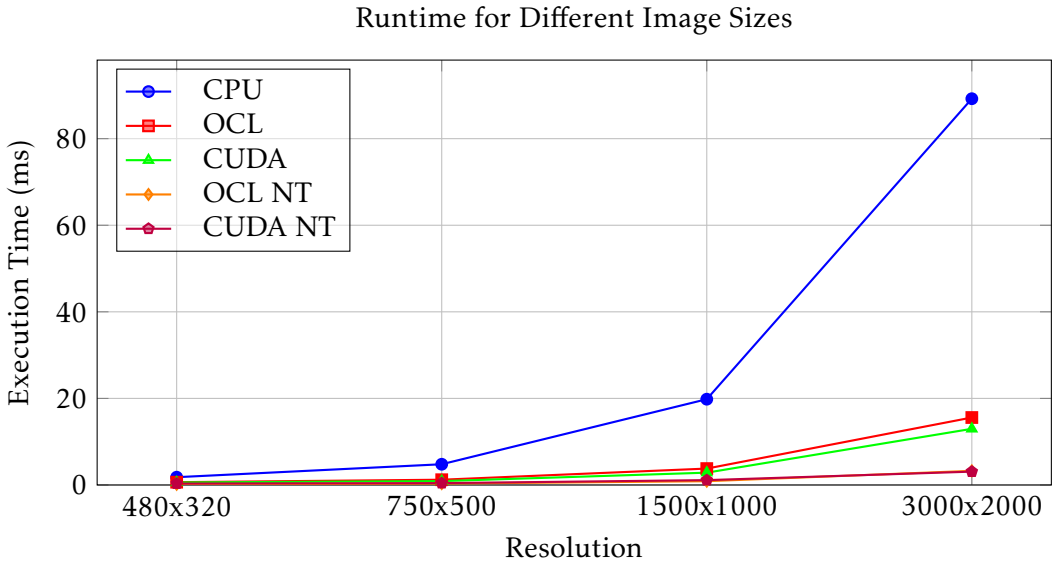


Figure 5.8: Chart of FFT runtime (ms) with increasing image size.

In summary, the move to a GPU is clearly recommended for frequency domain operations. A more detailed comparison of FFT and convolution filter performance on GPU can be found in the work of Fialka et al. [57].

5.2 Visual Results

The bicycle2 image from the MDB-Q dataset was chosen as an example. Additionally, a modified variant with added Gaussian noise was created as shown in Figure 5.9. In Figure 5.10 to Figure 5.14 the edge responses of most methods from Table 5.1 and Table 5.2 are visualized.

The Roberts method uses a 2x2 operator and produces the sharpest results with high sensitivity to noise (see next section for more details). The resulting edge region is shifted by 0.5 pixels in both axes, and the image has one row and one column less. The derivative methods with the same kernel size (Prewitt, Sobel, Scharr, Gauss, Poisson) give quite similar results. The smaller the kernel size, the sharper the edge response, but also more sensitivity to noise. Larger kernels produce smoother results, reducing the response to noise and high frequency texture (see Figure 5.15). The steerable filter produces a more blurred response and is sensitive to noise, but can detect lines (see spokes on a bicycle).

The other quadrature-based methods (SQF LG/PO and PC) and the Susan method can also detect lines. For PC, the response is often decreasing at junctions, but in many cases stable even for low contrast edges (see the horizontal door line on the right in the grayscale image of Figure 5.9). It also seems to handle noise quite well. The response of the image with noise seems almost clearer, since a better value for the noise reduction can be estimated. But on closer inspection, the response is more perforated. The Susan operator also provides a fairly sharp response, but also contains much more noise. The RMG looks quite similar to the Sobel operator, but provides less information with much longer runtime. The RCMG seems to have some amplified responses. The result is quite similar to the RMG, but some edge responses

Original bicycle2 | bicycle2 with noise

bicycle2 grayscale | bicycle2 grayscale with noise

FIGURE 5.9: Example image to visualize the edge response results.

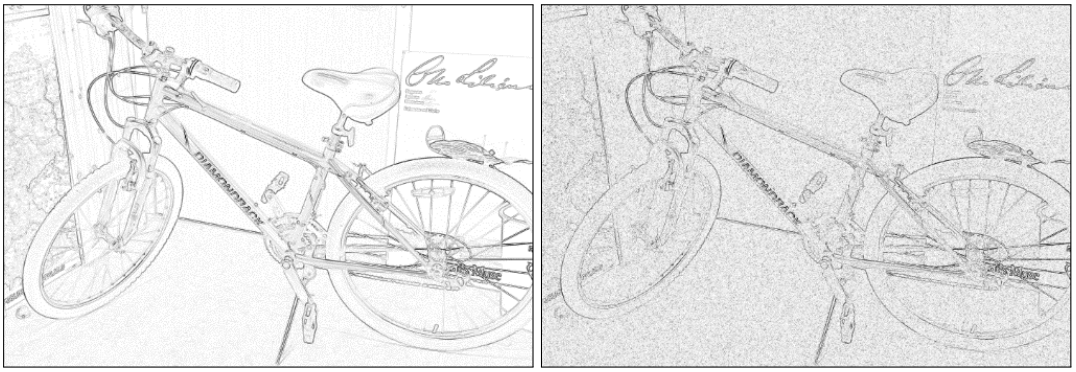
are more pronounced (compare the edge response of the door frame or the spokes), due to the inclusion of color information for processing. Noise sensitivity is quite high for all RMG variants and less pronounced for the RCMG method.

The responses of the Laplace operators appear weaker because they are zero crossing maps and the important information is the thin black line between two light lines. This behavior prevents the edge regions from being connected at junctions. The SQFF Laplacian response appears even weaker at many locations where the other methods provide a clear response. However, it seems to be much more stable to noise. The LoG gives quite sharp and clear responses (depending on the kernel size), with a high sensitivity to noise, but less strong than the simple Laplace operator.

5.3 Sensitivity to Noise

To estimate the noise sensitivity for the edge response methods, the average absolute differences are calculated based on the edge response of the original image and the edge response of the image with Gaussian noise added. The images are taken from the MDB-Q image dataset.

In Table 5.5 the errors for the different noise levels are listed. They are also visualized as a bar graph in Figure 5.17. In most cases the noise sensitivity increases almost linearly with increasing noise level. For the other methods, a logarithmic sensitivity can be observed. The most sensitive method seems to be the SUSAN operator, although it is logarithmic. It is followed by the Laplacian method and the Roberts



Roberts Method

Prewitt Method

Scharr Method

Sobel Method

FIGURE 5.10: Edge responses of bicycle2 image part 1 - Left: without noise, Right: with noise.

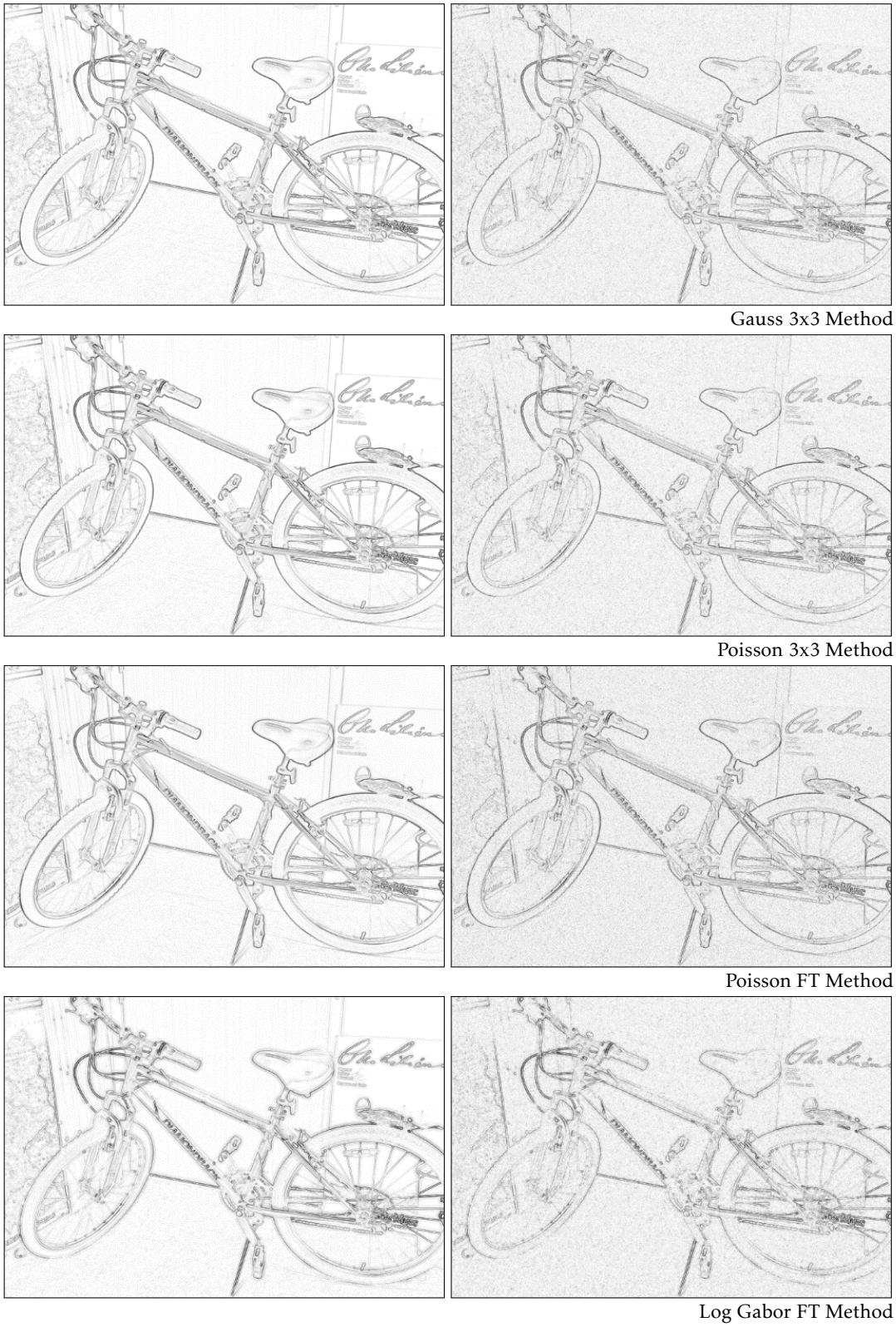


FIGURE 5.11: Edge responses of bicycle2 image part 2 - Left: without noise, Right: with noise.

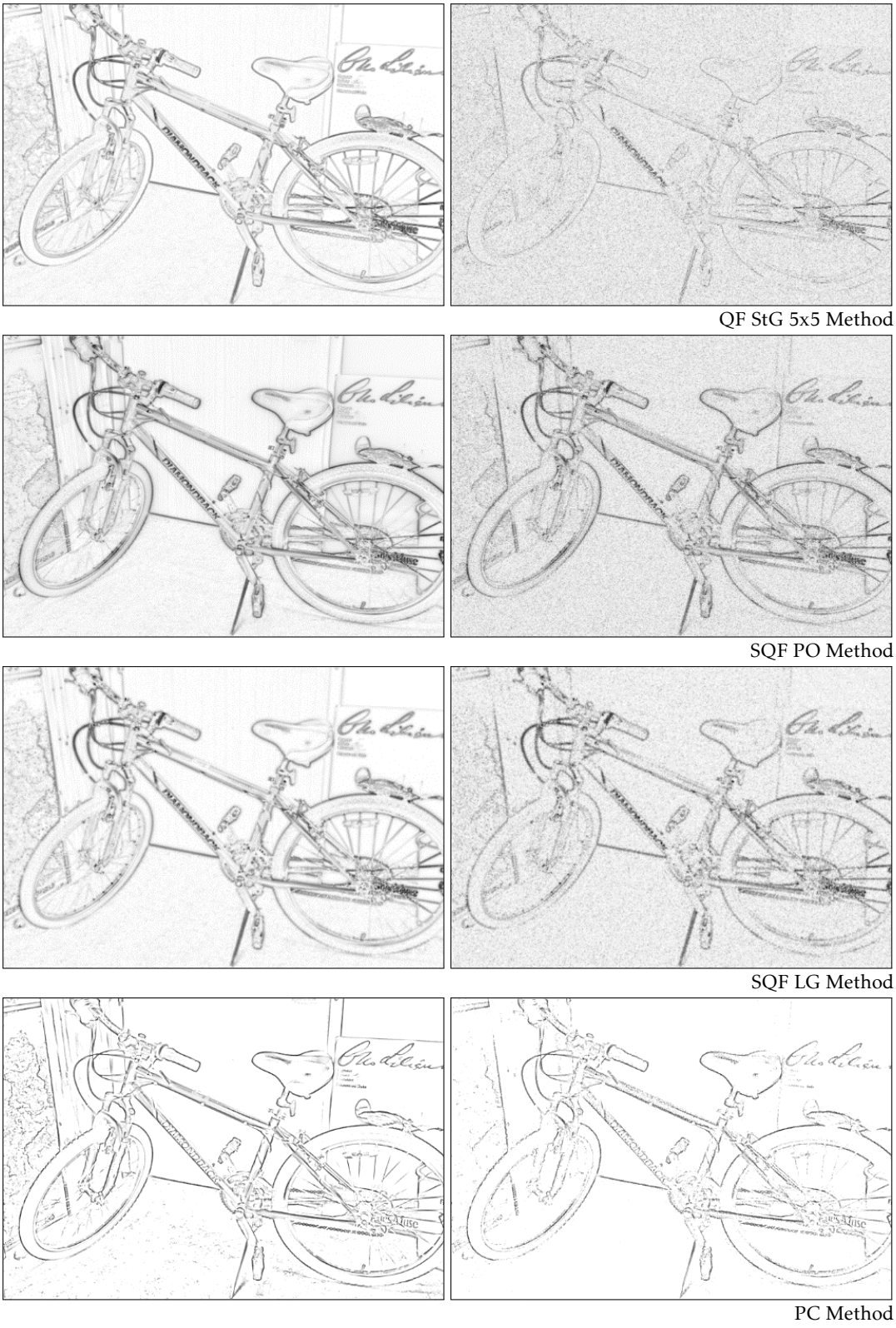


FIGURE 5.12: Edge responses of bicycle2 image part 3 - Left: without noise, Right: with noise.

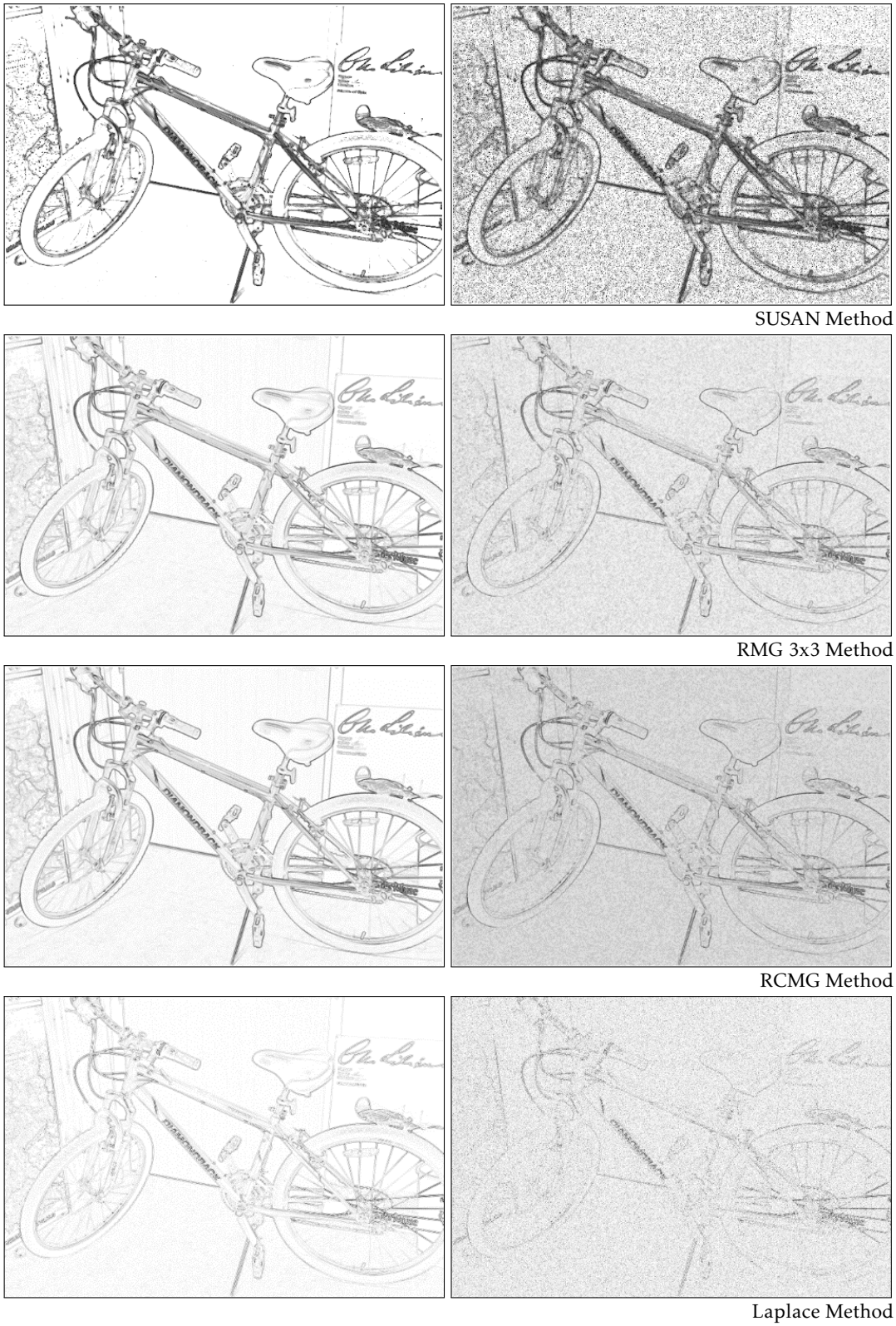


FIGURE 5.13: Edge responses of bicycle2 image part 4 - Left: without noise, Right: with noise.

Method	noise 10	noise 20	noise 30	noise 40	noise 50
Roberts (2x2)	0.065	0.136	0.206	0.273	0.336
Prewitt (3x3)	0.037	0.078	0.119	0.158	0.195
Scharr (3x3)	0.042	0.088	0.133	0.178	0.219
Sobel (3x3)	0.039	0.082	0.124	0.165	0.203
Gauss (3x3)	0.041	0.087	0.132	0.175	0.216
Gauss (5x5)	0.029	0.061	0.093	0.123	0.152
Gauss (7x7)	0.024	0.051	0.077	0.102	0.127
Gauss (9x9)	0.021	0.043	0.065	0.087	0.107
QF StG (3x3)	0.074	0.158	0.242	0.324	0.401
QF StG (5x5)	0.062	0.134	0.208	0.279	0.348
QF StG (7x7)	0.052	0.114	0.177	0.238	0.297
QF StG (9x9)	0.046	0.099	0.154	0.208	0.259
SQF Po (3x3)	0.063	0.135	0.208	0.278	0.345
SQF Po (5x5)	0.041	0.088	0.136	0.183	0.227
SQF Po (7x7)	0.036	0.076	0.118	0.158	0.196
SQF Po (9x9)	0.033	0.071	0.109	0.146	0.182
SQFF Lg	0.041	0.087	0.134	0.181	0.225
SQFF Po	0.046	0.102	0.160	0.219	0.274
PC	0.027	0.039	0.045	0.048	0.050
Susan (37)	0.088	0.304	0.453	0.534	0.582
Susan (3x3)	0.119	0.368	0.511	0.589	0.635
RMG (3x3)	0.056	0.121	0.187	0.253	0.315
RMG (5x5)	0.056	0.126	0.199	0.271	0.341
RCMG (3x3)	0.009	0.031	0.065	0.110	0.162
LoG (3x3)	0.095	0.187	0.275	0.359	0.437
LoG (5x5)	0.073	0.145	0.213	0.278	0.339
LoG (7x7)	0.059	0.116	0.170	0.222	0.271
LoG (9x9)	0.050	0.099	0.145	0.189	0.231
Laplace SQFF	0.016	0.036	0.060	0.087	0.117

Table 5.5: Average absolute differences between original image and noisy images (noise 10, 20, 30, 40 and 50)

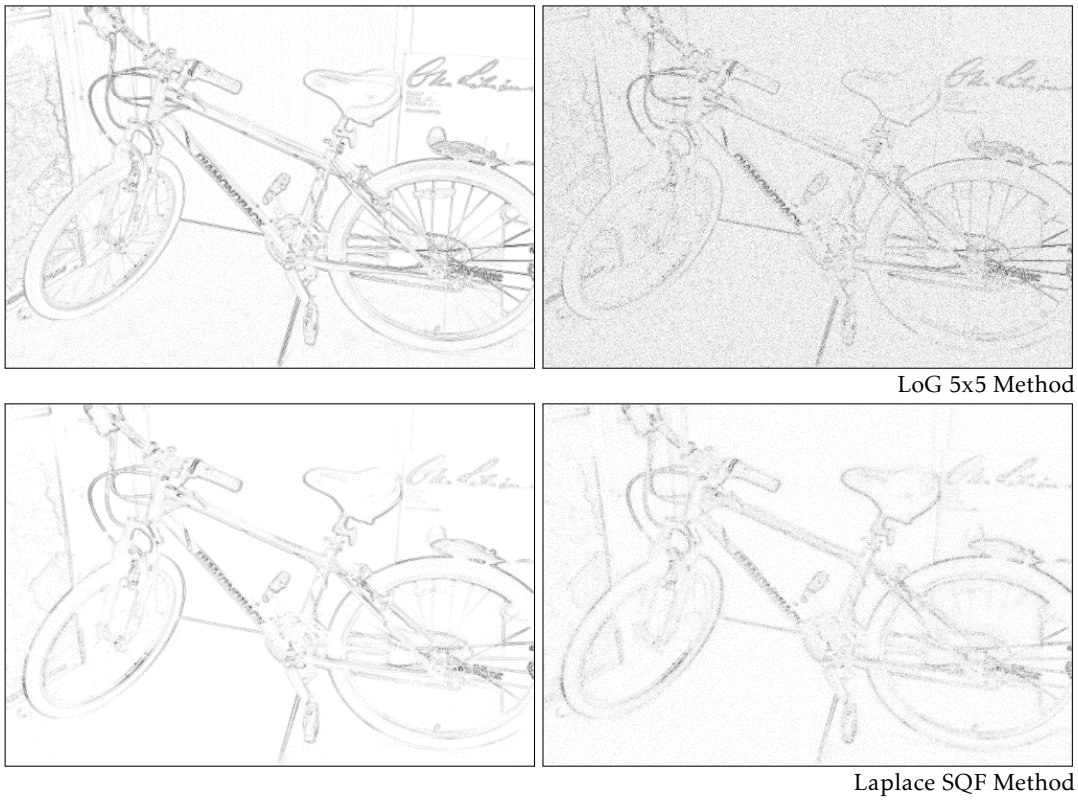


Figure 5.14: Edge responses of bicycle2 image part 5 - Left: without noise, Right: with noise.

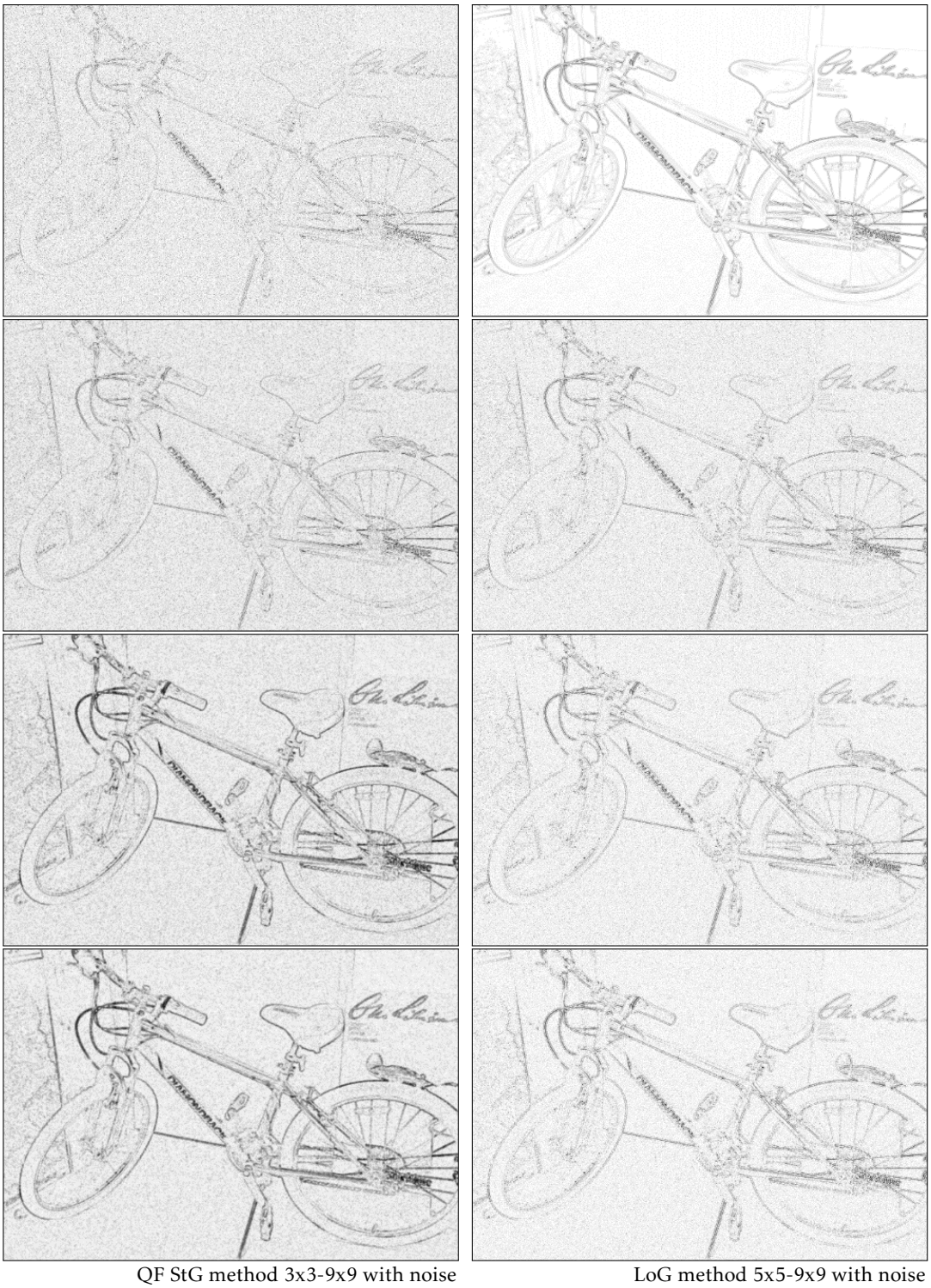
operator, QF StG and RMG methods. The RCMG seems to be more robust than the RMG and performs slightly better than the derivative methods (Prewitt, Scharr, Sobel, Gauss, SQF PO, SQFF LG, and SQFF PO). The best result is obtained with the PC method. This is due to the noise suppression and scale space used in this method. It is closely followed by the Laplacian-like response estimated from the phase derivative operator Laplace SQFF.

The graph in Figure 5.18 illustrates the behavior of noise sensitivity with increasing filter kernel size. Larger kernels reduce noise sensitivity, but also rapidly increase computation time.

5.4 Gradient Orientation Precision

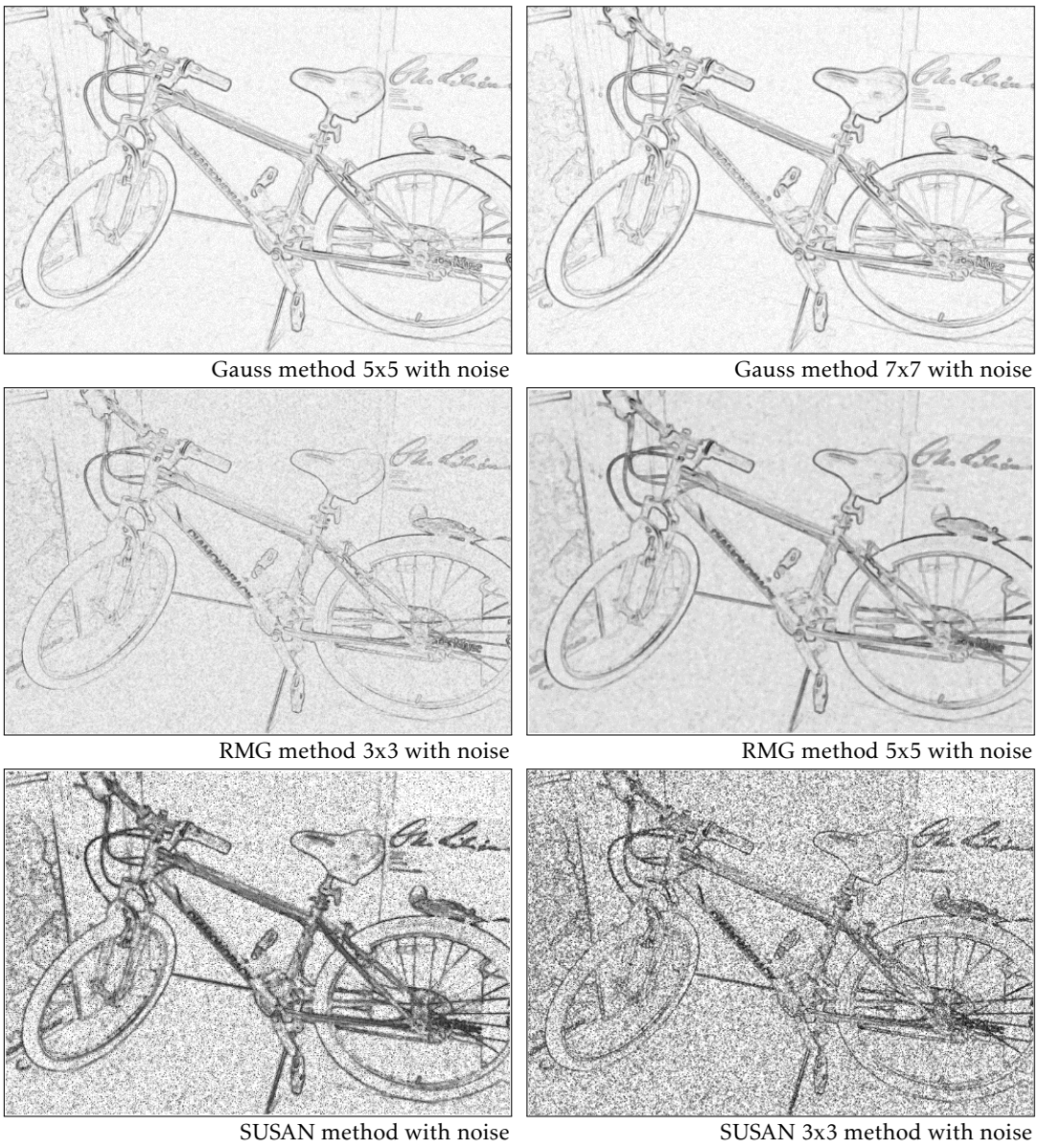
Most of the presented edge response methods can provide orientation information, which is needed for many post-processing steps such as non-maximum suppression (see subsection 6.2.2). Depending on the method, the accuracy of the orientation data varies greatly. Two different setups were used to examine the orientation data provided:

- The box setup uses a simple light gray box on a dark gray background that is rotated up to 45° in 1° steps as shown in the top left of Figure 5.19. A mask is created to select the important areas at the box edges to compare it with the ground truth edge orientation. The corner areas are removed to measure only stable regions (see top right of Figure 5.19). The ground truth map is created by applying the appropriate orientation to each of the four edge regions of the



QF StG method 3x3-9x9 with noise LoG method 5x5-9x9 with noise

FIGURE 5.15: Illustration of edge responses using different kernel sizes part 1 - Left column: QF StG method, Right column: LoG method.



Gauss method 5x5 with noise

Gauss method 7x7 with noise

RMG method 3x3 with noise

RMG method 5x5 with noise

SUSAN method with noise

SUSAN 3x3 method with noise

FIGURE 5.16: Illustration of edge responses using different kernel sizes part 2.

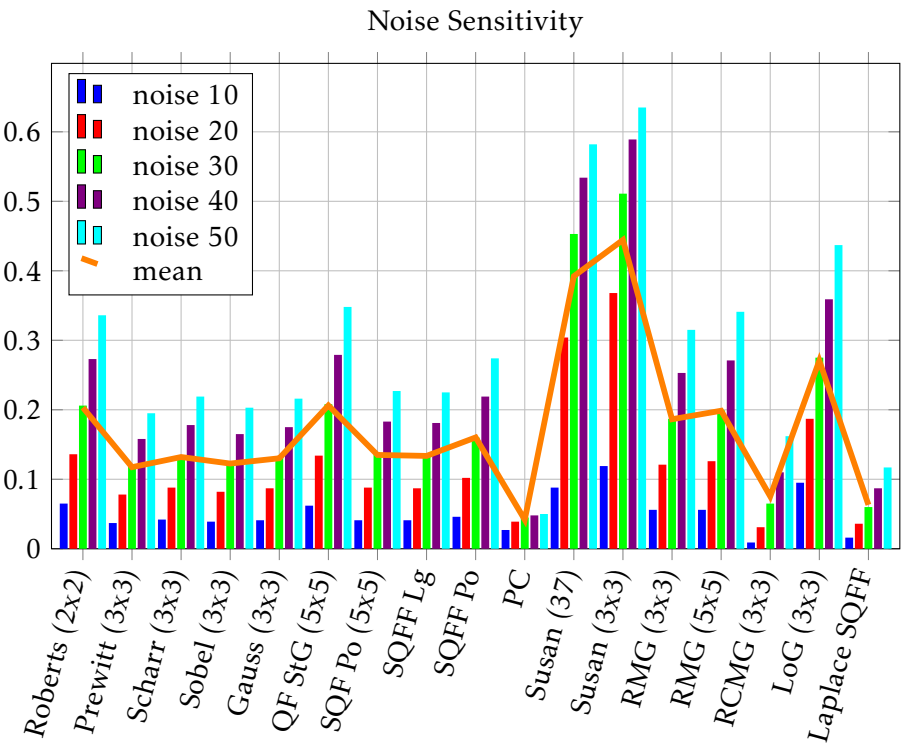


FIGURE 5.17: Average absolute differences between original image and noisy images (noise 10, 20, 30, 40 and 50)

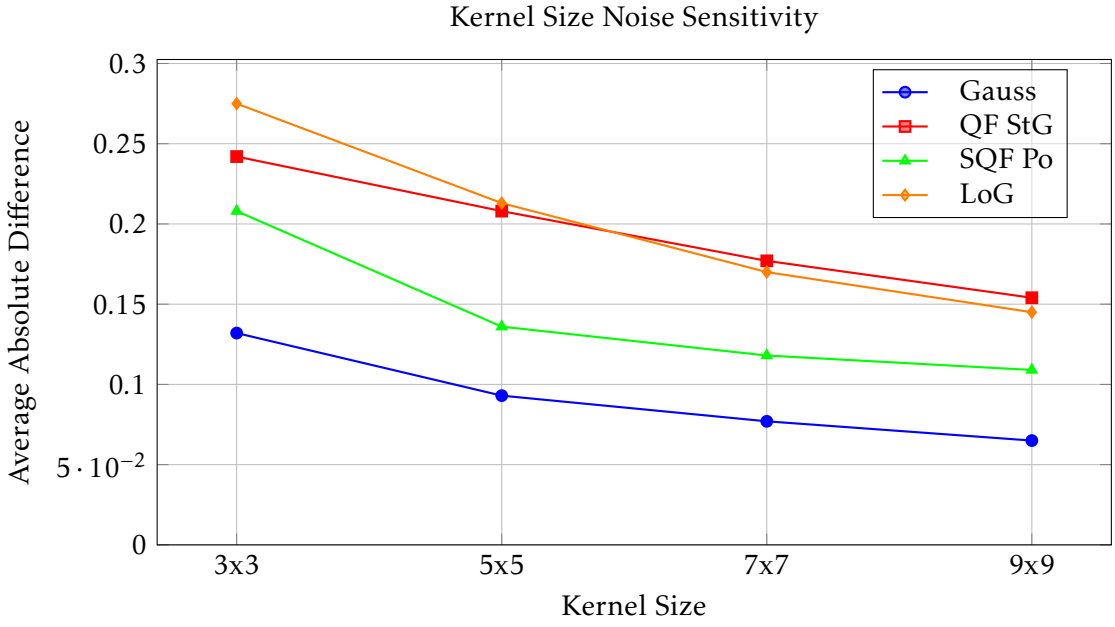


FIGURE 5.18: Noise sensitivity behavior as filter kernel size increases.

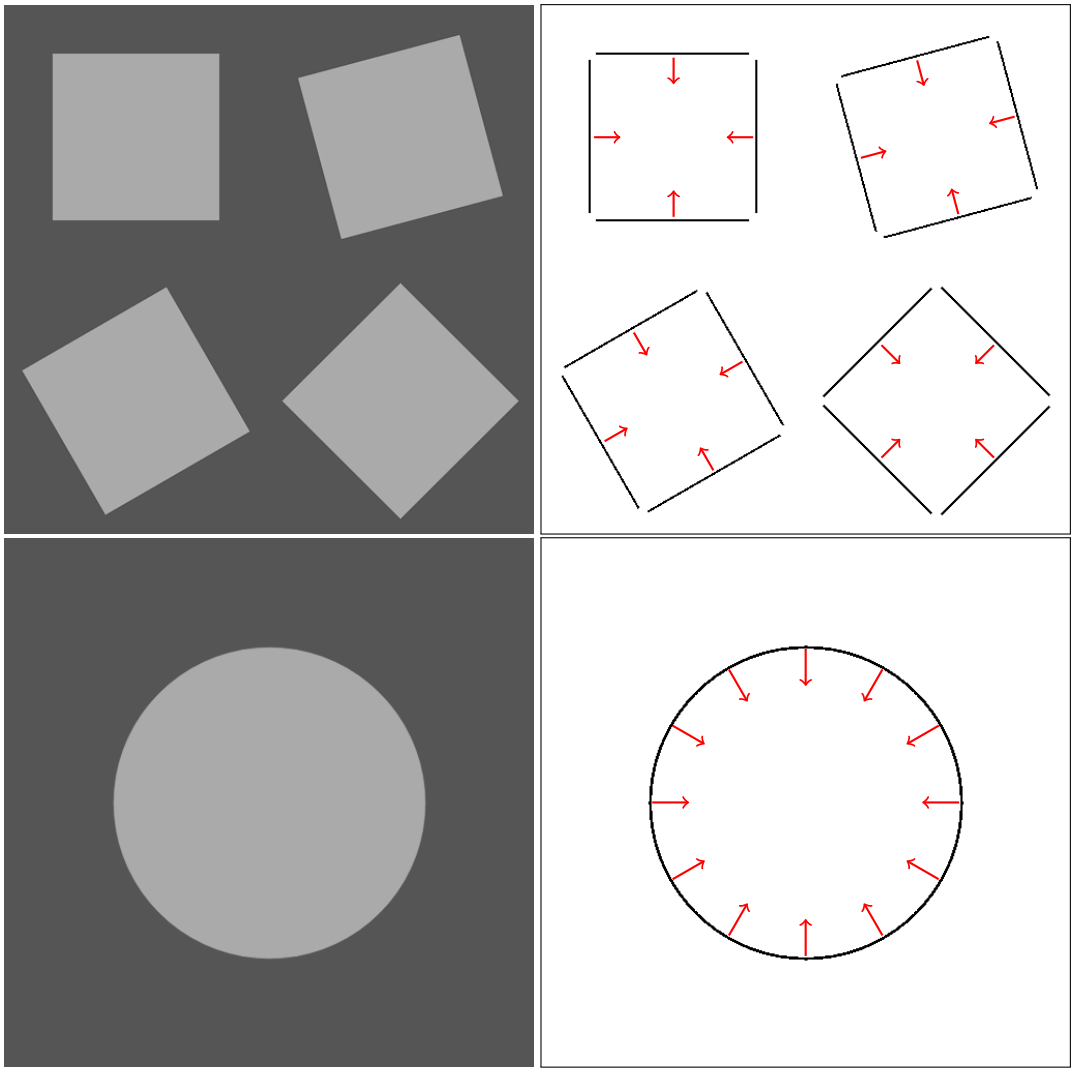


FIGURE 5.19: Top left: Box images with rotations of 0°, 15°, 30°, and 45°. Upper right: Selection mask for box images. Bottom left: Disk image. Bottom right: Disk image selection mask. The red arrows indicate the direction of the ground truth gradient.

box. The width of the regions is estimated from the Sobel gradient magnitude of the original box image.

- The disk setup uses a light gray filled disk drawn on a dark gray background (see bottom left of Figure 5.19). From the known geometry, the orientation for each pixel can be computed and is used as ground truth. Again, a mask is used to select the important edge region of the disk with a width derived from the magnitude of the Sobel gradient, as shown in Figure 5.19 on the bottom right.

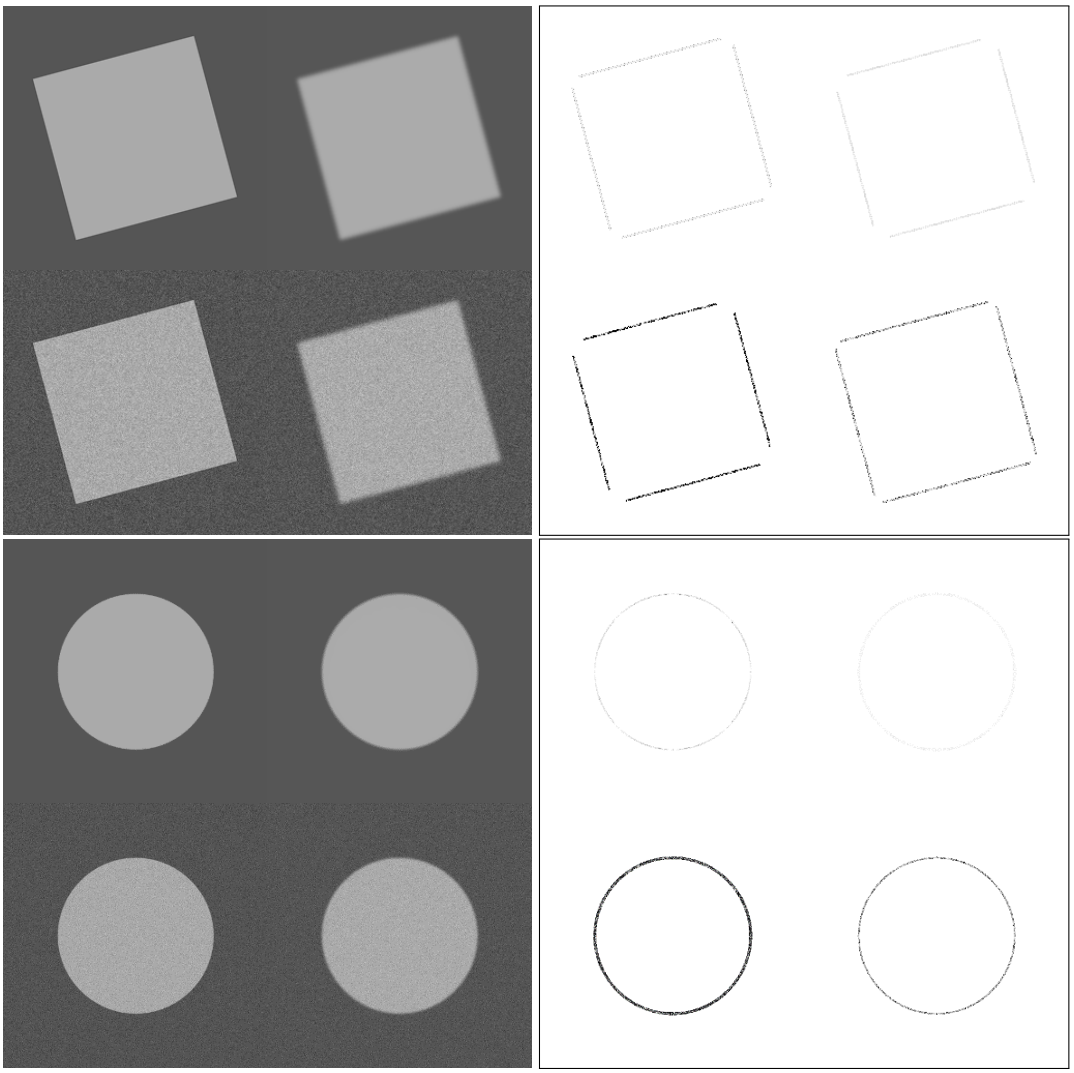


Figure 5.20: Left: The four test variants for the box and disk (normal, smooth edges, noise, blur and noise). Right: Example visualization of orientation error using the Scharr operator.

Both setups compute the mean orientation error in radians. In addition, a blurred, a noisy, and a combined variant of the setup are included in the test (see Figure 5.20 on the left). All methods are configured to use double precision (64 bit) for calculations. This prevents the results from being distorted by different calculation accuracies.

An enhanced visualization of the orientation error is shown in Figure 5.20 on the right. For the box setup, the error depends strongly on the rotation of the box. For zero rotation and a rotation of 45° the error is lowest for all methods. In the example

Method	none	blur	noise 10	b+n 10	noise 40	b+n 40
Susan (37)	0.502	0.445	0.613	0.713	1.245	1.402
Susan (3x3)	0.580	1.598	0.819	1.016	1.434	1.556
RMG (3x3)	1.067	0.776	1.038	1.120	1.322	1.554
RMG (5x5)	1.311	1.105	1.042	0.893	1.231	1.461
Prewitt (3x3)	0.108	0.030	0.220	0.436	0.818	1.768
Scharr (3x3)	0.048	0.030	0.221	0.514	0.933	1.936
Sobel (3x3)	0.039	0.029	0.198	0.458	0.852	1.830
Sobel (5x5)	0.024	0.016	0.115	0.229	0.469	1.019
Sobel (7x7)	0.018	0.012	0.088	0.155	0.350	0.665
Sobel (9x9)	0.015	0.009	0.074	0.121	0.294	0.500
Gauss (3x3)	0.041	0.029	0.216	0.504	0.935	1.923
Gauss (5x5)	0.017	0.019	0.139	0.293	0.577	1.331
Gauss (7x7)	0.013	0.014	0.113	0.220	0.459	0.997
Gauss (9x9)	0.011	0.012	0.094	0.174	0.381	0.749
QF StG (3x3)	0.184	0.353	0.285	1.334	1.189	1.573
QF StG (5x5)	0.094	0.329	0.171	1.080	0.917	1.562
QF StG (7x7)	0.079	0.290	0.144	0.891	0.766	1.530
QF StG (9x9)	0.072	0.252	0.128	0.701	0.650	1.488
SQF Po (3x3)	0.031	0.029	0.204	0.478	0.872	1.875
SQF Po (5x5)	0.027	0.018	0.134	0.268	0.548	1.228
SQF Po (7x7)	0.024	0.015	0.115	0.218	0.463	0.972
SQF Po (9x9)	0.022	0.014	0.108	0.195	0.433	0.857
SQFF LG	0.032	0.069	0.271	1.249	1.139	2.612
SQFF Po	0.118	0.230	0.216	0.428	0.814	1.604
PC	0.029	0.035	0.120	0.218	0.474	0.957

TABLE 5.6: The values represent the mean gradient orientation error in radians for the box setup. Each column represents a different test setting: none: original image; blur: Gaussian smoothing with $\sigma = 2$; noise 10/40: Gaussian noise with an intensity of 10/40; b+n 10/40: Gaussian smoothing and noise.

Method	none	blur	noise 10	b+n 10	noise 40	b+n 40
Susan (37)	0.457	0.477	0.502	0.439	0.827	0.894
Susan (3x3)	0.624	0.773	0.694	0.652	1.038	1.030
RMG (3x3)	0.656	0.964	0.873	0.877	1.005	1.055
RMG (5x5)	0.643	1.073	0.788	0.730	0.916	0.993
Prewitt (3x3)	0.059	0.011	0.120	0.224	0.481	0.866
Scharr (3x3)	0.036	0.011	0.141	0.258	0.560	0.968
Sobel (3x3)	0.031	0.010	0.122	0.232	0.499	0.924
Sobel (5x5)	0.017	0.009	0.067	0.114	0.264	0.530
Sobel (7x7)	0.014	0.008	0.047	0.074	0.184	0.332
Sobel (9x9)	0.012	0.008	0.040	0.065	0.159	0.243
Gauss (3x3)	0.033	0.011	0.130	0.254	0.560	0.966
Gauss (5x5)	0.018	0.009	0.080	0.144	0.335	0.641
Gauss (7x7)	0.015	0.009	0.064	0.113	0.249	0.526
Gauss (9x9)	0.014	0.008	0.052	0.089	0.207	0.400
QF StG (3x3)	0.100	0.199	0.319	0.977	0.875	1.091
QF StG (5x5)	0.061	0.176	0.215	0.874	0.778	1.006
QF StG (7x7)	0.046	0.152	0.162	0.774	0.721	1.022
QF StG (9x9)	0.047	0.130	0.145	0.648	0.617	1.000
SQF Po (3x3)	0.029	0.011	0.124	0.241	0.532	0.939
SQF Po (5x5)	0.021	0.009	0.076	0.142	0.326	0.620
SQF Po (7x7)	0.019	0.008	0.065	0.108	0.244	0.487
SQF Po (9x9)	0.018	0.008	0.060	0.094	0.223	0.434
SQFF LG	0.035	0.025	0.203	0.662	0.715	1.275
SQFF Po	0.044	0.058	0.113	0.191	0.464	0.812
PC	0.016	0.008	0.065	0.111	0.269	0.502

TABLE 5.7: The values represent the average gradient orientation error in radians for the disk setup. Each column represents a different test setting: none: original image; blur: Gaussian smoothing with $\sigma = 2$; noise 10/40: Gaussian noise with an intensity of 10/40; b+n 10/40: Gaussian smoothing and noise.

a rotation of 15° is visualized. Only the error visualization of the Figure 5.20 gives a good impression of how sensitive the gradient orientation is to noise. Adding noise to the image significantly increases the error, which is even more pronounced when applied to primitives with smooth edges.

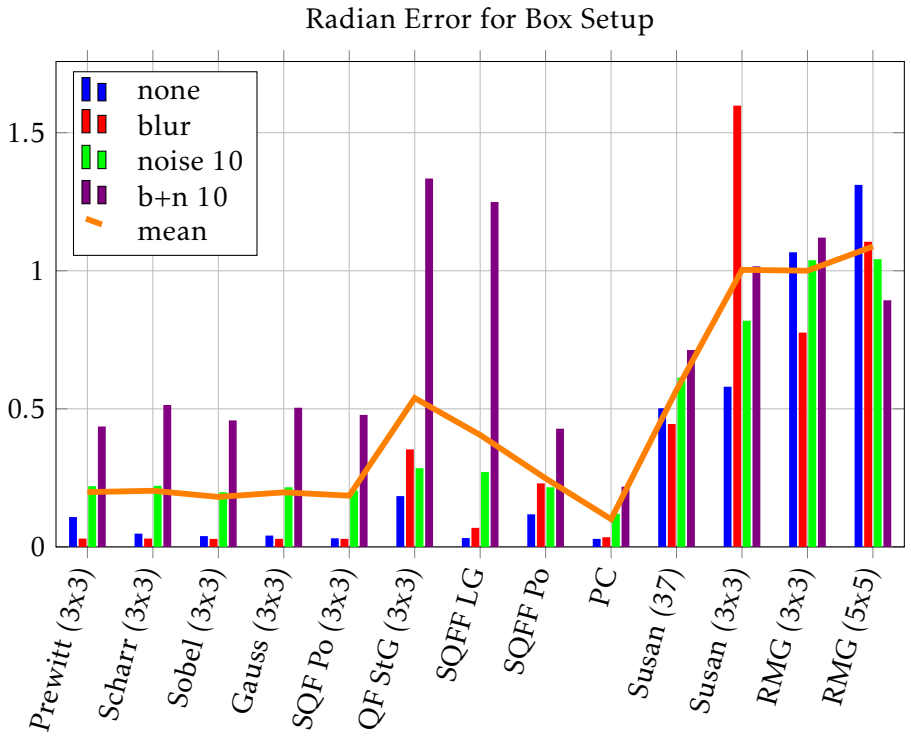


FIGURE 5.21: Bar chart for the orientation errors of box setup containing the four test variants (normal, blurred, noise 10 and both) and the average error.

It is obvious from Table 5.6 and Table 5.7 that not all methods are equally sensitive to noise. It is even more obvious when looking at the plots of Table 5.6 and Table 5.7, which show the errors for the box and disk setup grouped by method. The results show that all derivative-based methods perform quite similarly. They can be very accurate and perform best on soft edges (blur), but are quite sensitive to noise, especially on smooth edges. The more complex QF StG methods appear to be less accurate and have the opposite response to blur than the other derivative methods. The sensitivity to noise can be reduced by increasing the kernel size, as can be seen for the Sobel, Gaussian, QF StG, and SQF PO methods. For all four methods, the larger kernel size results in a lower error rate, as shown in Figure 5.17.

The edge response methods that operate in the frequency domain show a different behavior. While the Log Gabor variant seems to be quite accurate without noise, it produces large errors with added noise. The Poisson variant is less accurate without noise, but also less sensitive to noise. The phase congruency methods seem to be more robust to noise (due to the use of a scale space) and perform quite well for all categories. The Susan and RMG methods are less sensitive to noise than the other methods. However, the accuracy of these methods is generally very poor. Only if a rough orientation estimation is needed, the data can be considered.

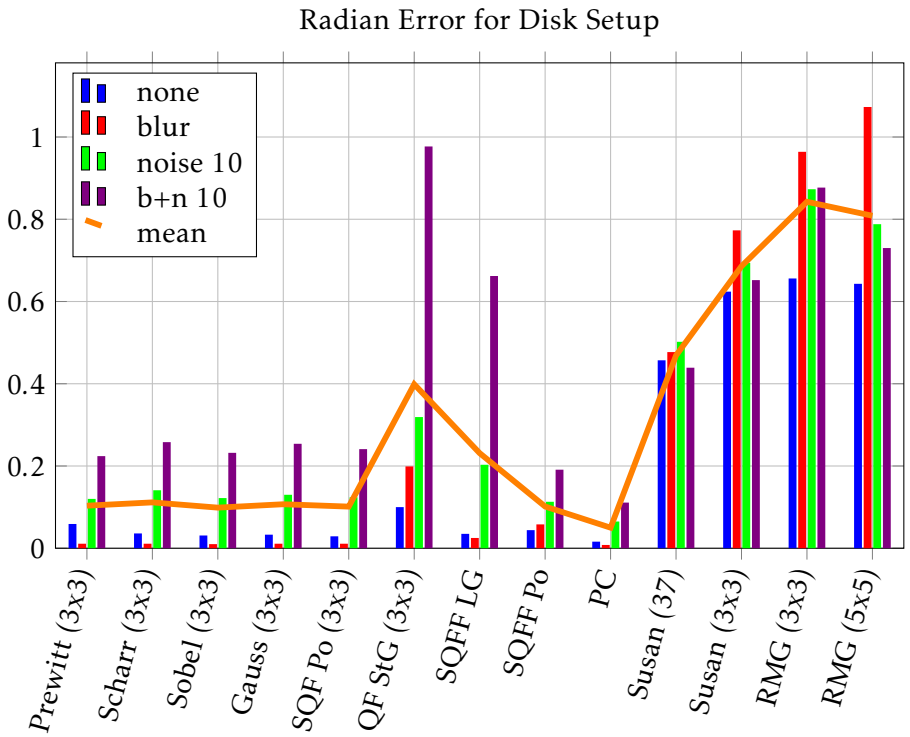


FIGURE 5.22: Bar chart for the orientation errors of disk setup containing the four test variants (normal, blurred, noise 10 and both) and the average error.

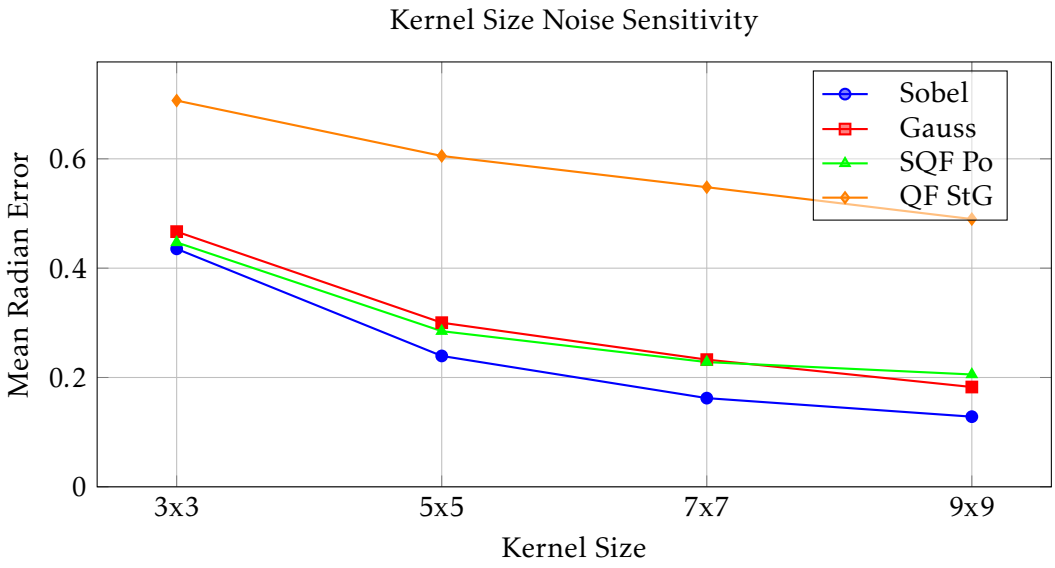


FIGURE 5.23: Shows the mean radian error of both setups using all variants depending on the filter kernel size.

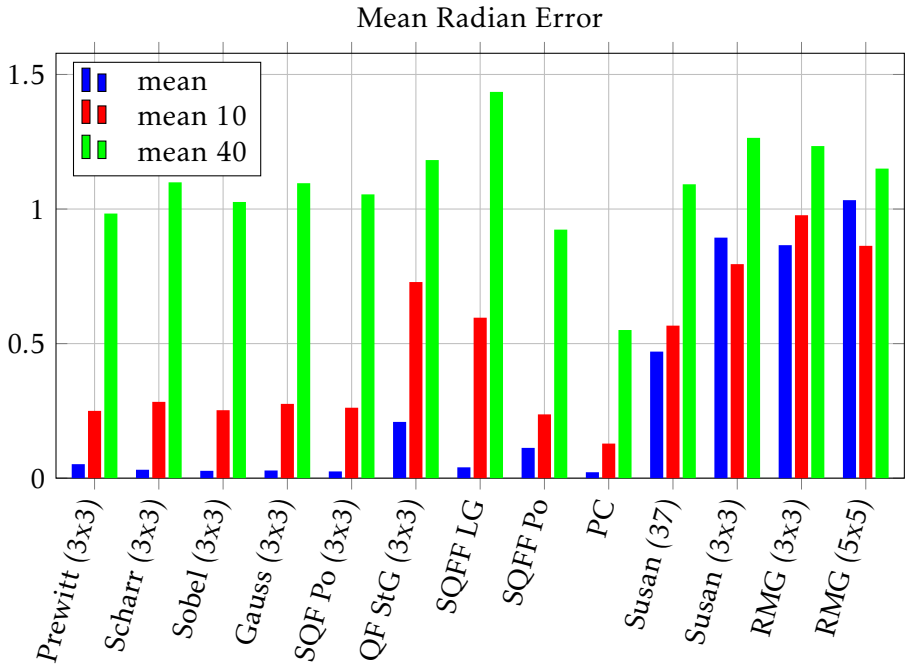


FIGURE 5.24: Mean error for no noise, images with low noise (mean 10) and high noise (mean 40). The mean error is based on both test setups (box and disk), both variants (no blur, with blur).

The plots show that both test setups produce similar orientation errors, making the results more reliable. Figure 5.24 visualizes the combined average error of both test setups as a bar chart. It gives a quick overview of what kind of orientation accuracy can be expected from different edge detection methods depending on the quality of the images used.

5.5 Discussion

In general, the classic gradient-based methods work quite well. They are fast and produce accurate results for most images. However, they are sensitive to noise and can produce false positives on smooth edges (depending on the kernel size). Most of the more complex methods don't provide much better results (at least for the general use case), but have a much higher processing time. An exception might be the PC method, which provides quite good accuracy with strong robustness to noise. The Laplacian approaches can also be very fast, but seem to be less accurate than the gradient equivalents and don't provide orientation information.

The take away is to stick with the fast gradient-based methods. Only for special tasks, such as processing a low-light image sequence with a high noise rate, precise reconstruction methods, or requirements for features that are not provided by the simple filters, consider using another method.

Chapter 6

Edge Region Operations

Once the edge responses have been generated using the methods discussed in the previous chapter, further processing steps can be applied to enhance or extract specific features from the edge regions. For example, the edge regions can be filtered by a threshold, thinned to final edge pixels, or even optimized to extract edge points with sub-pixel accuracy.

6.1 Threshold Methods

In image processing, thresholding methods are used for image segmentation. By converting a grayscale image into a binary image, they separate the image pixels as foreground pixels, which are the pixels of interest, and the background pixels [203] (see (A) of Figure 6.1). The segmentation can be modified to have more than the two segments for foreground and background. However, in this case, the thresholding method is used to separate the pixels with significant edge responses from the pixels with weak and noisy edge responses, making two segments the optimal choice.

6.1.1 Global Threshold

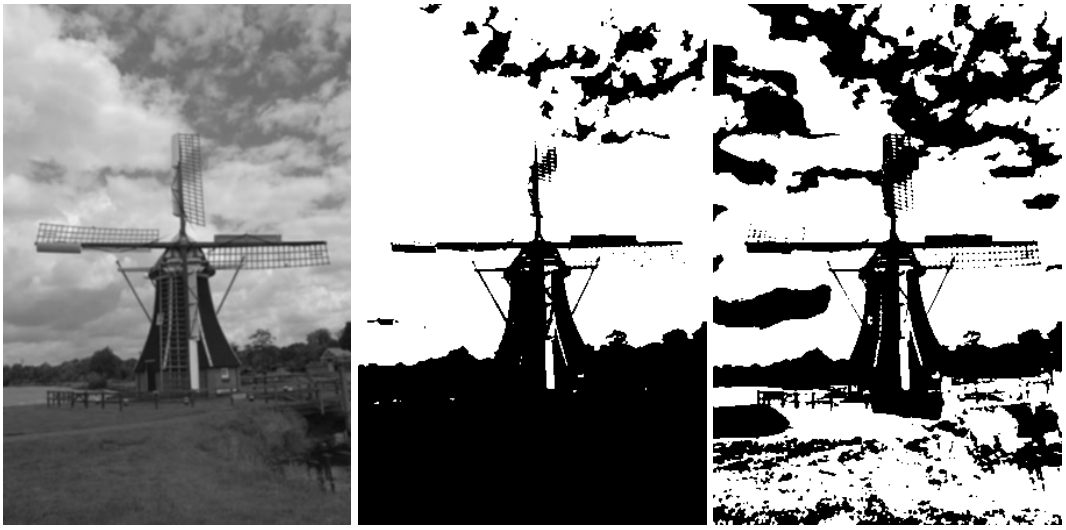
The simplest method to extract the edge and corner pixels, is to apply a global threshold function $T_g(m)$ to the given edge response E. All values above a given threshold t are accepted, all others are rejected (see (B) of Figure 6.1):

$$T_g(E) = \left\{ \begin{array}{ll} 0 & if\ E < t \\ 1 & if\ E \geq t \end{array} \right\} \tag{6.1}$$

If a low threshold is chosen, edge pixels with a low signal will be extracted but also the response of noise. A high threshold will reduce the detection of noise edge pixels, but may miss some important edge data (see Figure 6.2). The optimal threshold has to be determined experimentally for every image or derived from the signal to noise ratio. See subsection 6.1.4 for more information about the estimation of thresholds.

6.1.2 Double Thresholds

One problem that can occur with the global thresholding method is streaking. If the edge response fluctuates above and below the threshold, a broken edge line will appear. One method to drastically reduce the likelihood of streaking is to use double thresholds. If an edge segment's value is above the upper threshold, it is accepted immediately. If the value is below the lower threshold, it is immediately rejected. Values between the upper and lower thresholds are accepted if they are connected to a value that is already accepted. This method is called hysteresis and is already used in fields such as electronics and biology. Canny first proposed this method for edge



(A) Original image (left) with global threshold applied (center) and dynamic threshold (right).

(B) Sobel magnitude image (left) with global threshold applied (center) and dynamic threshold (right)

FIGURE 6.1: Threshold example using the Otsu threshold estimator (see subsection 6.1.4).

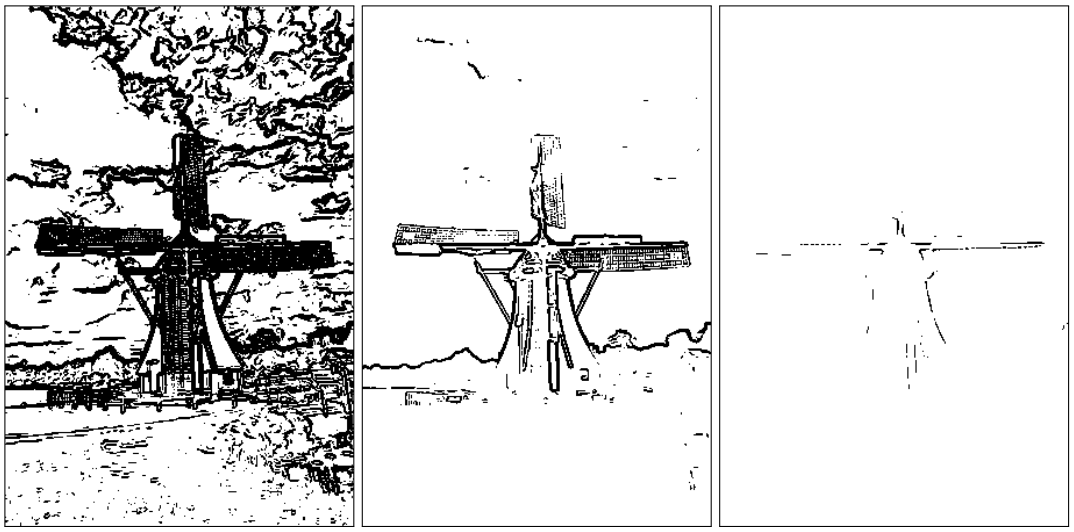


FIGURE 6.2: Threshold example using low, good and high threshold values.

detection [31]. He recommends using a ratio of upper to lower threshold in the range of two or three to one, based on the predicted signal to noise ratio.

A disadvantage of this method is the additional computational cost. The global threshold method can be applied independently to each pixel, which is optimal for parallelization via SIMD[1] or GPU[2] based implementations. The connectedness constraint of hysteresis is difficult to parallelize and will be the bottleneck of the algorithm [138].
Usually, this method is incorporated into most edge collection algorithms, as discussed in the next chapter (see chapter 7). A general implementation is not provided in this thesis.

6.1.3 Local Thresholds

The local thresholding method is applied to image regions. Each region R_i has its own threshold t_i, which can be chosen independently (see Figure 6.3 left image as example):

$$T_l(x,y) = \left\{ \begin{array}{ll} 0 & if\ E(x,y) < t_i \\ 1 & if\ E(x,y) \geq t_i \end{array} \right\} \forall (x,y) \in R_i \tag{6.2}$$

This approach can be useful for images with variations in illumination, such as high and low contrast regions that produce strong and weak edge responses. The use of hysteresis with two thresholds is also recommended, since the streak effect can also occur in image regions. This method increases the problem of finding appropriate thresholds. An automatic method to find a good threshold is preferable. However, this increases the computational cost.

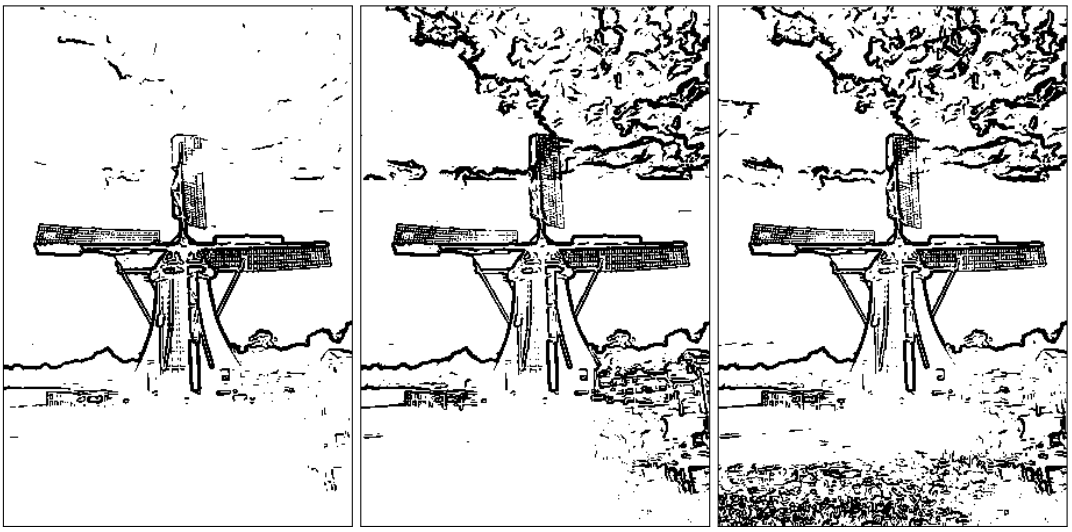


FIGURE 6.3: Threshold examples using a global threshold method (left), a local threshold method (center, 3x3 tiles) and dynamic threshold (right, radius for threshold estimation: 50 pixels). All methods used the Otsu threshold estimator.

Dynamic Thresholds

The dynamic thresholding method uses a maximized local thresholding method that applies a separate threshold to each pixel in the input data (see Figure 6.3 right image as example):

$$T_d(x,y) = \left\{ \begin{array}{ll} 0 & if\ E(x,y) < t(x,y) \\ 1 & if\ E(x,y) \geq t(x,y) \end{array} \right\} \tag{6.3}$$

The threshold for a pixel is usually determined by analyzing the neighboring pixels. This approach is robust to changes in illumination, but results in even higher computational cost.

6.1.4 Threshold Estimation

To prevent the extraction of edge pixels caused by sensor noise, compression artifacts, or weak illumination changes, a threshold is used in the edge detection process. It can also be used to find only the strongest responses, but in most cases all relevant edge data is preferred. If edge extraction is coupled to a camera, the noise level can be estimated or learned for the device [131], otherwise general denoising [28] or image smoothing approaches can be used before edge detection is applied. See section 3.3 for more information on this topic.

After preprocessing, it is still necessary to choose an appropriate threshold for edge extraction. For a fixed set or special category of images, an optimal threshold can be determined empirically. Another approach is to choose a very low threshold, at the risk of extracting noise and artifacts, and to perform the elimination of falsely detected edges in some post-processing steps, such as accepting only edge segments with a minimal number of supporting points or analyzing the edge point orientation distribution for a given segment.[2]. The third approach is to derive the threshold

[1] Single Instruction, Multiple Data (SIMD), https://en.wikipedia.org/w/index.php?oldid=666309006

[2] Graphics Processing Unit (GPU), https://en.wikipedia.org/w/index.php?oldid=663296163

from image properties, for example by analyzing the signal-to-noise ratio [233, chapter 6] in the response image.

Canny [31] proposes to use a histogram on the filtered image to determine the edge responses caused by noise. The edge responses of noise differ from the responses of real edges in that they have a Gaussian-like distribution centered near zero, while the responses of real edges are stronger and more sparsely distributed. The noise should be visible in the histogram as a high count in the lower intensity region. The threshold is then chosen from the histogram as the intensity above the first peak in the lower intensity region.
The threshold selection from histograms can be optimized by using the thresholding method of Otsu [164]. The algorithm assumes two classes of pixels in an image (foreground and background pixels) and calculates the optimal threshold to separate the two classes by a minimum combined spread or within-class variance. The method can be efficiently implemented by using a histogram and the fact that the threshold with the minimum within-class variance also has the maximum between-class variance.

Fang et al. [235] combined Otsu's method with Canny edge detection by estimating the upper threshold for Canny by applying Otsu's algorithm to a given grayscale image before applying Canny. The lower threshold is calculated as 0.5 times the upper threshold.

This approach only makes sense if the given image can be separated into foreground and background without losing significant edges. A more general approach is to apply Otsu's method to the edge response image to estimate the upper threshold and choosing the lower threshold by using Canny's selection criteria. In this case, either the response image intensity range must be scaled down to a reasonable range that can be expressed by a histogram, or Otsu's method must be adapted to use values (e.g. count pairs instead of a histogram). This method can be used to determine a global, local, or dynamic threshold, as discussed in the previous sections (also see Figure 6.1 and Figure 6.3).

6.2 Edge Region Thinning

To reduce the extracted edge regions to final thin edges, a thinning process is required. A simple approach is to use morphological operators applied to the edge regions. A more advanced approach, which also uses the edge orientation, is the non-maximum suppression (NMS) method. NMS can be combined with a zero crossing (ZC) method to extract the edge regions. The ZC method can be used to extract the edge regions directly from the second derivative of the image.

6.2.1 Morphological Thinning

Thinning can be implemented as a morphological operation as shown in Figure 6.4. It applies several structuring elements to a binary image by a hit-and-miss transformation. To obtain a binary image of the edge responses, a global threshold can be applied as described in section 6.1.

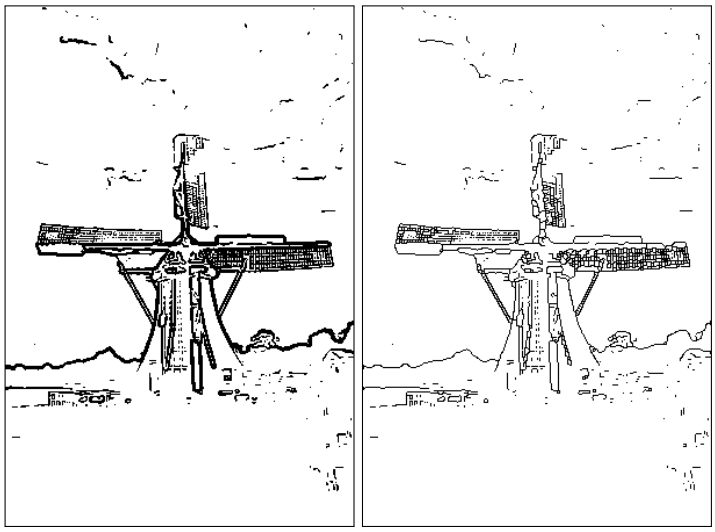


FIGURE 6.4: Example of an erosion operator applied (OpenCV implementation based on [237]) (right) to a binary edge response image (left).

The hit-and-miss transform is a slight extension of the erosion and dilation operations. The operator can contain foreground and background pixels in the structuring element (ones and zeros, with the meaning of zeros as background). Multiple texturing elements with different orientations must be applied to get the final result. The thinned edges may have some erroneous pixels called spurs. They can be reduced by applying pruning, which is the same process as thinning, but using other structuring elements [68].

Morphological operations can be performed quite efficiently on binary images, and they produce nicely connected edges even at junctions. The final thin edges are located in the center of the binary edge regions. However, the maximum edge response, which indicates the true edge location, is not necessarily at the center of the edge region. A more accurate approach is to search for the maximum edge responses and suppress the other responses.

6.2.2 Non Maximum Suppression

The concept of this approach is to use the edge energy response in combination with the edge orientation. In the case of a classical gradient, as introduced in section 2.3, the edge orientation is perpendicular to the gradient direction (see Figure 8.5). The magnitude of an edge pixel is compared to the magnitude of its two direct neighbors that are perpendicular to the edge orientation. If the magnitude is greater than the magnitude of the neighbors, it is marked as a real edge pixel (see Figure 6.5) as described in [31].

A fast implementation of NMS selects two pixels from the eight-connected neighborhood along the gradient direction to determine if the central pixel is a local maximum. This can be done by quantizing the gradient orientation into one of four principal directions, each corresponding to a 45° rotation: vertical (top–bottom), horizontal (left–right), and the two diagonals. This coarse approach is also applicable to methods that provide only a rough orientation estimate, such as the RCMG or SUSAN edge

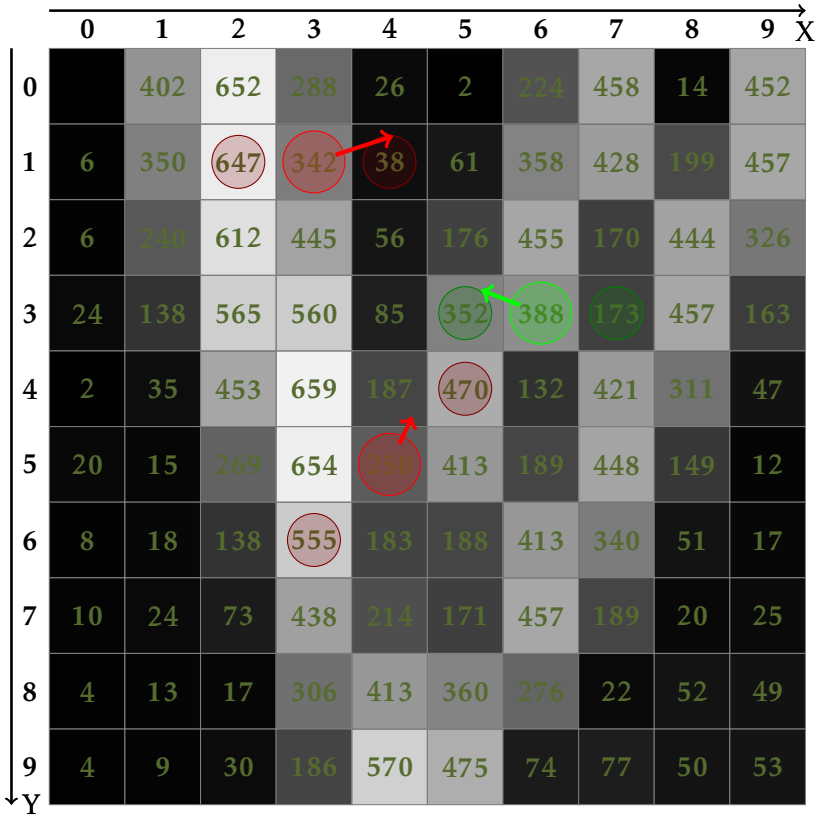


FIGURE 6.5: This figure shows a magnitude snippet of the windmill with two cases for NMS. Large green circle: Real maximum. The arrow shows the gradient direction (which is perpendicular to the edge orientation) and strength, the smaller darker circles show the selected neighbors. Large red circles: Suppressed edge response because there is no maximum between the neighbors selected based on the gradient.

detectors.

When a more accurate gradient orientation is available, a more precise NMS can be performed by interpolating the gradient magnitudes of neighboring pixels instead of simply comparing the two discrete neighbors. This is achieved by sampling the magnitude values along the gradient direction (perpendicular to the edge) at sub-pixel positions, using interpolation methods such as bilinear or bicubic interpolation. This approach better handles "edge cases," where the true maximum does not align exactly with integer pixel coordinates.

In the example shown in Figure 6.5, the fast variant compares the magnitudes of the two discrete pixels determined by the quantized gradient direction (illustrated by the smaller circles). The more precise variant instead interpolates the magnitude at the exact positions indicated by the gradient vector (arrow), leading to more accurate suppression decisions (see Figure 6.9 for an interpolation example).

Since each pixel can be checked independently, NMS is a local problem and can be efficiently computed using parallel processing methods [158].

A main problem of NMS is that branching edges get corrupted, creating gaps in the thinning process. This happens because in this case the direction of an edge is ambiguous, since there are multiple orientations. Therefore, the estimated orientation does not necessarily correspond to the real edge orientation, resulting in the wrong neighbors being chosen for the non-maxima evaluation. Another problem occurs when the width of an edge region is very large and the maxima are poorly localized. This can cause an edge to be shifted by more than one pixel, resulting in gaps again.

These problems can be reduced by using the slower NMS approach, which interpolates the size of neighboring pixels. However, multiple orientations can still occur, making the direction of an edge ambiguous. An approach to reduce unwanted gaps in combination with fast NMS, especially for junctions, has been proposed by Ding et al. [44]. They distinguish between major edges, which are computed by the fast NMS method, and minor edges, which have a local maximum in one of the four main directions. The major edge map is a subset of the minor edge map. The minor edge map contains some missing data, especially at junctions, that is not present in the major map. But it also contains false edges. To separate the minor true edges from the false edges, the authors traced the minor edge contours and kept only those parts of the minor edge traces that were connected to major edges. Additional attention must be paid to false loops. Since this approach must find the connected edge components to trace the edges, the local processing for each edge pixel is lost.

Another early approach to solving the junction problem was introduced by Li et al. [127]. He also observed the instability of gradient orientation at junctions. He stated that real free endpoints from the NMS process can be easily distinguished from false free endpoints by analyzing the gradient in the edge direction. If the gradient is increasing and a peak occurs in about 3σ (depending on the image smoothing) pixels, it is a false free end. To continue the end, his algorithm continues the edge based on the gradient until it hits an already identified edge pixel, and additionally searches back (about 2σ) to check for a non-isolated segment and consistent orientation strength. Rothwell et al. [182] used a topology-based method with an adapted Tsao-Fu thinning algorithm [216] to extract the junctions. For junction refinement, he

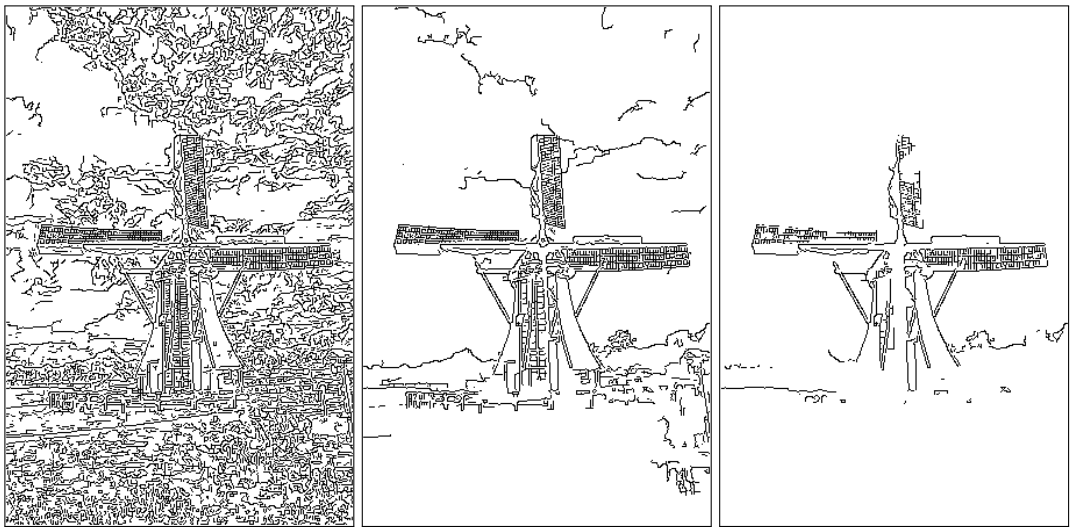


FIGURE 6.6: NMS examples using double threshold (hysteresis) to extract the final edge pixels. Left: Uses a low threshold pair. Center: Uses a threshold pair estimated via the Otsu method. Right: Uses a high threshold pair.

suggested fitting a parametric model similar to Blaszka et al.'s method [18], which could also be used with corner detection. Another approach to recover and fit junctions was proposed by Maire et al. [144]. They use lines that interpolate edge segments and compute the intersection of them to find well localized junctions. A more expensive approach is to use high order steerable filters [62] to find multiple edge directions or some kind of orientation optimization [55, section 6.1] that gives a good orientation estimate of the strongest direction.

To solve the problems with NMS, either some kind of connected components method or fitting model is required. Finding connected components is introduced in chapter 7, including methods to close gaps or fix junctions in the segmentation process. The methods are similar to the approaches of Ding et al. and Li et al. [44, 127]. In chapter 8 approaches for fitting line segments to the extracted edge segments are introduced. They can be used to optimize the junction locations similar to the approach of Maire et al. [144].

6.2.3 Zero Crossing

The edge location of second derivative filters, such as the Laplacian operator designed by Marr & Hildreth [147], can be found at the zero crossing of the edge response signal. To extract the zero crossings, a threshold at zero can be used to process a binary image where all values < 0 are marked as background and all values >= 0 are marked as foreground. The boundaries between foreground and background represent the locations of zero crossings and can be extracted with morphological operations (see subsection 6.2.1) by marking every foreground point that has at least one background neighbor.

To avoid foreground or background bias, points on both sides of the threshold should be considered, by selecting the pixel that is closer to zero and hopefully closer to the

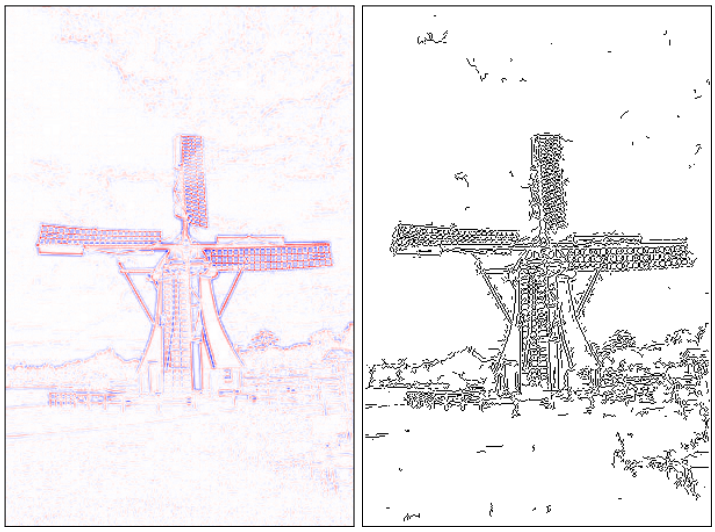


FIGURE 6.7: Laplace and ZC example. Positive values of the Laplace are red, negative blue. For the ZC image on the right, the threshold was estimated with some adapted Otsu estimator for Laplace responses.

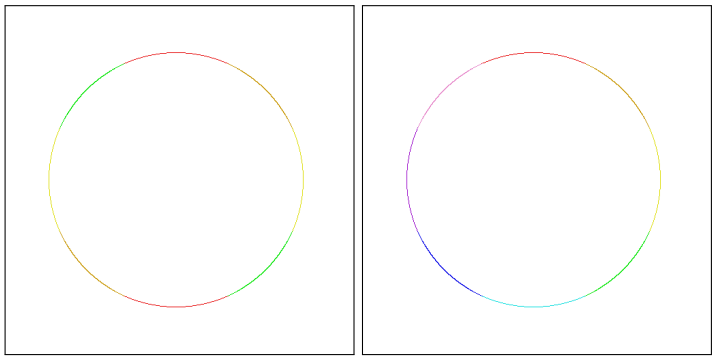


FIGURE 6.8: ZC of a disk image. The edge pixels are colored by a 4 region map (left image using fast4 zc) and an 8 region map (right image using fast8 zc).

real intersection.

The resulting edge map does not necessarily consist of perfect thin edge segments. However, efficient isotropic filters, which are commonly used to extract the edge maps, do not provide orientation information. Therefore, edge thinning must be done by orientation-independent methods, such as morphological thinning, or at least a rough orientation map must be estimated (e.g., in a separate step) to apply NMS-like approaches (see Figure 6.8 for ZC with orientation information).

One problem with this approach is that zero crossings occur not only at real edges. In regions of the image where the intensity of the gradient is not stable and fluctuates near zero, false edges are detected. Instead of simply testing for < or >= 0, the absolute distance of a pair of zero crossings can be calculated and used to determine a kind of edge strength that is used for thresholding (see section 6.1 and Figure 6.7 as example).

6.3 Sub-Pixel Precision

To find more accurate edge locations, some form of pixel interpolation must be applied. A simple approach can be realized by upscaling the responses, e.g. by a factor of three, using bilinear or bicubic interpolation methods and then applying non-maximum suppression (NMS, see subsection 6.2.2). The resulting edge pixels can be used directly for further processing or downscaled independently to obtain sub-pixel accurate localization. However, this approach is very expensive and not always suitable for post-processing steps, e.g. if the post-processing can't handle overrepresented pixel locations. A solution to this problem could be to collect multiple locations of a pixel depending on the orientation and then use the average to calculate the final location.

Other approaches have been performed by locally fitting different curve models according to the NMS process [42, 10], by approximating the ERF function [81], or by using a moment based edge detection approach [141, 64, 105]. Canny [31] suggested using second order derivatives to find the zero crossing in combination with the gradient direction, which was addressed by Rothwell et al. [182]. A more general version of this method was described by Lindenberg [129]. The main advantage of this approach is that good subpixel localization can be achieved by linear interpolation (see the next section for more details).

While Canny's proposed method is quite practical for methods that need to compute a second derivative, such as quadrature-based filters, a more general approach is to use an interpolation method for NMS. In the case of NMS, since the sub-pixel precision calculation is quite simple and can be processed independently for each pixel, it can be done very efficiently. Bailey stated that, depending on the interpolation method, the precision can be up to 1% [10] of the width of a pixel. It can be easily integrated into the processing pipeline with NMS and is therefore an available option for most edge region detectors (for more details on this approach, see subsection 6.3.1). For corner or junction regions, the orientation used for interpolation is not reliable. In this case, the interpolation should be skipped or the orientation should be estimated from the neighborhood of connected stable edge pixels.

6.3.1 Subpixel Accuracy for NMS (Devernay Approach)

This section introduces the basic Devernay approach to improve the accuracy of the final edge responses after applying NMS or ZC. It fits a parabolic model to the edge response of the NMS process. The parabolic model is defined as a quadratic function, which is a good approximation of the edge response in the neighborhood of the maximum.

In the process of estimating the NMS edge pixels, the magnitude is already calculated and stored. It is then used for the suppression test to perform the quadratic interpolation based on the three magnitude values along the orthogonal direction of the edges fitted to the parabolic model. The location is then estimated by the maximum of the interpolated quadratic curve:

$$\begin{aligned} f(x) &= ax^2 + bx + c \\ f'(x) &= 2ax + b \end{aligned} \tag{6.4}$$

To find the maximum of the curve, the first derivative must be zero:

$$2ax + b = 0 \;\rightarrow x = \frac{-b}{2a} \tag{6.5}$$

Substituting the magnitude values for the three known points, the result is:

$$m_- = a - b + c, \;\; m_0 = c, \;\; m_- = a + b + c \tag{6.6}$$

Then it is possible to compute the location of the maximum:

$$x = \frac{m_+ - m_-}{4m_0 - 2(m_+ + m_-)} \tag{6.7}$$

If the distance between the two outer points and the center point is 1 pixel, then the range for the maxima is limited to the range $[-0.5, 0.5]$, which is located at the center point. To obtain the final location, the result of x must be added to the pixel location depending on the gradient direction θ or the orthogonal direction of the edge pixel:

$$e = e_{NMS} + \begin{bmatrix} \cos(\theta)\, x \\ \sin(\theta)\, x \end{bmatrix} \tag{6.8}$$

Devernay [42] analyzed the resulting precision of the location in terms of the magnitude norm. Using a squared norm reduces the precision, so he suggested using the real norm by taking the square root.

Bailey [10] extended this approach with several models: the pyramid model for linear interpolation, the center of gravity (CoG) model, and the Sobel filter model. The models are based on three values m_-, m_0, m_+, where m_0 is the center pixel and m_- and m_+ are the left and right neighboring pixels. The models are defined as follows

$$\begin{aligned} x_{pyramid} &= \frac{m_+ - m_-}{2(m_0 - \min(m_-, m_+))} \\ x_{CoG} &= \frac{m_+ - m_-}{m_0 + m_+ + m_- - 3\min(m_-, m_+)} \\ x_{Sobel} &= \frac{m_+ - m_-}{2m_0} \end{aligned} \tag{6.9}$$

He compared the results of the line and step-edge detection models, taking into account the influence of noise and quantization. The parabolic method (basic approach) gave the best results in most cases. For thin lines, the CoG method outperformed the parabolic method.

To use the result in further processing like segmentation, a float representation of the location is not always advantageous. A simple solution is to store the difference of pixel and sub-pixel location for each edge point separately.

6.3.2 Subpixel Accuracy for ZC

A more accurate approach to estimating the edge pixel location is to use an interpolation similar to the concept introduced in subsection 6.3.1 (see Figure 6.9 for more details). In this case, a simple linear interpolation will give good results, since the curve is quite steep at the zero crossings and is well approximated by a linear line

near the zero. The sub-pixel location is calculated as:

$$\boldsymbol{e}_z = \frac{\boldsymbol{e}_j L\left(\boldsymbol{e}_j\right) - \boldsymbol{e}_i L\left(\boldsymbol{e}_i\right)}{L\left(\boldsymbol{e}_j\right) - L\left(\boldsymbol{e}_i\right)} \tag{6.10}$$

Where $\boldsymbol{e}_z \in \mathbb{R}^2$ is the final edge location in sub-pixel precision, based on the two adjacent edge pixels $\boldsymbol{e}_j$, $\boldsymbol{e}_i \in \mathbb{N}^2$ with a sign change in the Laplacian response function $L(x)$. If a good estimate of the gradient orientation θ of $\boldsymbol{e}_i$ is known (e.g. by calculating the first derivatives), $\boldsymbol{e}_j \in \mathbb{R}^2$ can be determined as:

$$\boldsymbol{e}_j = \boldsymbol{e}_i + \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \end{bmatrix} \tag{6.11}$$

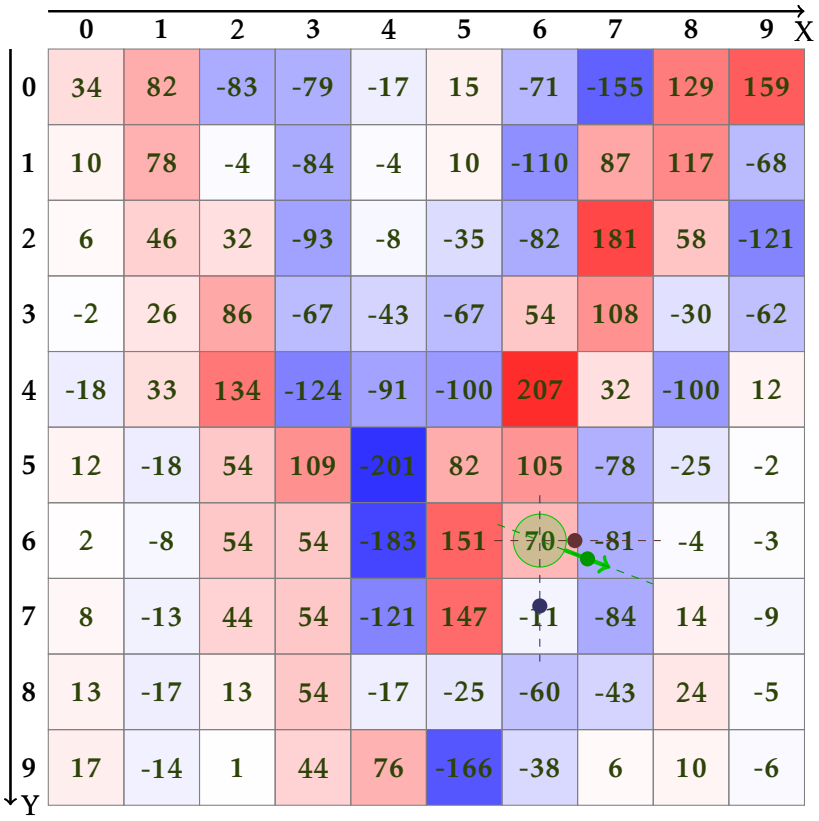


FIGURE 6.9: Visualization of zero crossings in the Laplacian response. Red cells represent positive values, blue cells represent negative values, and a selected zero-crossing pixel is marked as green circle with its gradient direction. The dashed lines indicate interpolation lines while the filled dots represent the interpolated zero crossing position. Dark red: Linear interpolation along x-axis. Dark blue: Linear interpolation along y-axis. Dark green: Bilinear interpolation along gradient.

By interpolating the Laplace response at $\boldsymbol{e}_j$, the location can be determined even more precisely.

6.4 Results and Discussion

In the first part of the analysis, a brief review of the thresholding methods is given to gain a better understanding of their behavior. However, the other two methods presented in this chapter are much more interesting, as they are the starting point for many further processing steps in edge extraction. Also the optimization steps to get more precise results are considered.

6.4.1 Threshold Methods

In general, only the double threshold method is used in combination with edge collection or edge flooding methods. The Otsu estimator can also be useful to find a meaningful threshold for further processing steps. Applying global, local or dynamic thresholding methods in combination with morphological operators would also be an option to extract final edge pixels. However, this approach is inferior to the other methods and is therefore rarely used.

Nevertheless, let's take a closer look at the methods and briefly study their runtime and behavior. The tests were performed on a high performance HP ZBook 15 Fury G8 notebook with 32GB RAM, Intel Core I7 CPU (11th Gen i7-11850H @ 2.50GHz, 8 Cores) and NVIDIA RTX A3000 Laptop GPU, running ubuntu 22.04 (jammy). It was applied only to the windmill image, which has a size of 300x448 pixels. The runtime was measured in milliseconds.

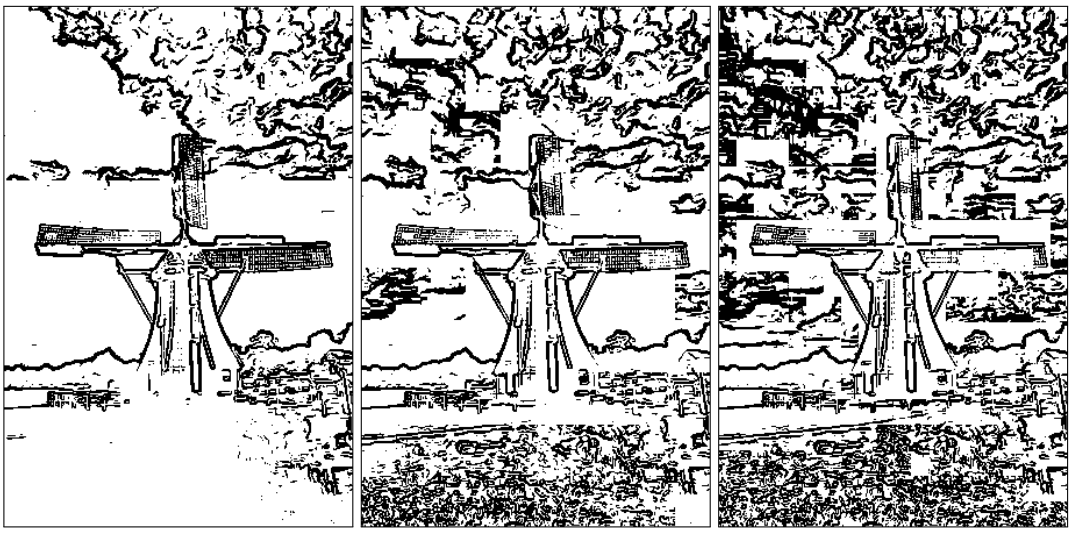
Method	Runtime (ms)
global img	0.368
global mag	0.476
global mag single threaded	0.278
local 3x3	0.060
local 10x10	0.078
local 20x20	0.222
local 20x20 single threaded	1.220
dynamic r10	24.867
dynamic r20	60.752
dynamic r30	91.397
dynamic r40	141.713
dynamic r50	195.957
dynamic r50 single threaded	1,384.659

Table 6.1: Runtime for various thresholding methods. As expected, the more complex methods (local with tiles or dynamic) require more runtime. All methods are optimized for multi-threaded execution. The single-threaded runtimes are marked accordingly.

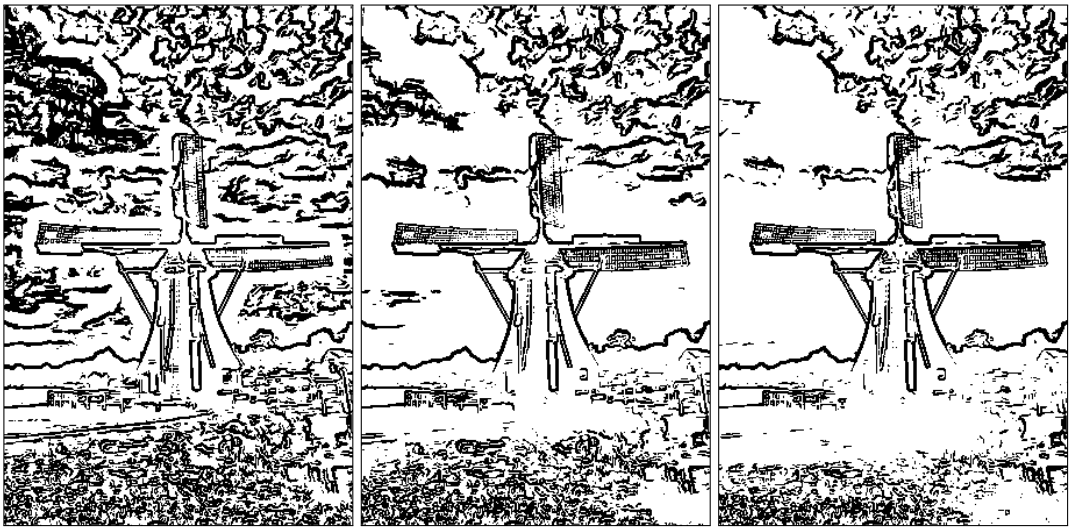
The single-threaded results of Table 6.1 show that the global threshold method is the fastest. The local threshold method is slower, but the runtime can be reduced by using larger tiles. The dynamic threshold method is the slowest because it requires calculating the Otsu threshold for each pixel. The runtime can be reduced by using a smaller radius or window, since the Otsu method takes less time to compute. However, due to the small radius, only a very small immediate neighborhood is considered,

which means that the threshold is determined extremely locally and accordingly few edge points are discarded (see (B) of Figure 6.10). Similar behavior can be observed for the local thresholding method with small tiles (see (A) of Figure 6.10).

While both methods provide similar results, the tiling method is much faster than the dynamic method. However, it is much more robust against artifacts. This can be clearly seen in Figure 6.10, where the tiling method sometimes produces strong artifacts at the tile boundaries, e.g. clearly visible in the windmill flights, which do not occur in the results of the dynamic method.



(A) Examples of local thresholding. The edge response image is divided into tiles, and for each tile the Otsu threshold is estimated and applied. Left: 3x3 tiles, Center: 10x10 tiles, Right: 20x20 tiles.



(B) Examples of dynamic thresholding. For each edge response point a separate Otsu threshold (based on a given radius) is estimated and applied. Left: Radius of 10, Center: Radius of 30, Right: Radius of 50.

FIGURE 6.10: Threshold examples using the Otsu threshold estimator (see subsection 6.1.4).

The parallelized execution of the three thresholding methods shows a slightly different picture regarding the fastest execution. Here, the global method is even slower due to the fact that the main runtime is caused by the Otsu estimator, which is not

parallelized. The application of the threshold to the image is parallelized, but is already highly optimized by the OpenCV operation, i.e. the runtime is longer due to the overhead of parallel execution. However, the runtime gain is significant for both the local and dynamic methods. For the local variant with 20x20 tiles, the speedup is about 5.5 times, while for the dynamic method with a radius of 50, it is up to 7 times faster.

6.4.2 NMS and ZC Results

In section 6.2 several approaches to edge region thinning were introduced. The main goal is to reduce the edge regions to a single pixel line. A common approach is to use morphological operators. But they lack precision because they don't consider the strongest edge responses. Better methods are non-maximum suppression for first derivative edge responses and zero crossing for second derivative edge responses (Laplacian). Both methods extract the edge pixel along the "strongest" response, resulting in more accurate edge pixels. Interpolation methods can be used to further increase accuracy.

To compare the available methods in terms of accuracy and runtime, a small experiment was performed. An ultra-high resolution synthetic image was generated based on a ground truth polygon. The image was first smoothed with a Gaussian low-pass filter and then resampled to produce an accurate image downscaled by a factor of 100 (see the result in Figure 6.11). The polygon was also downscaled as sub-pixel accurate ground truth.

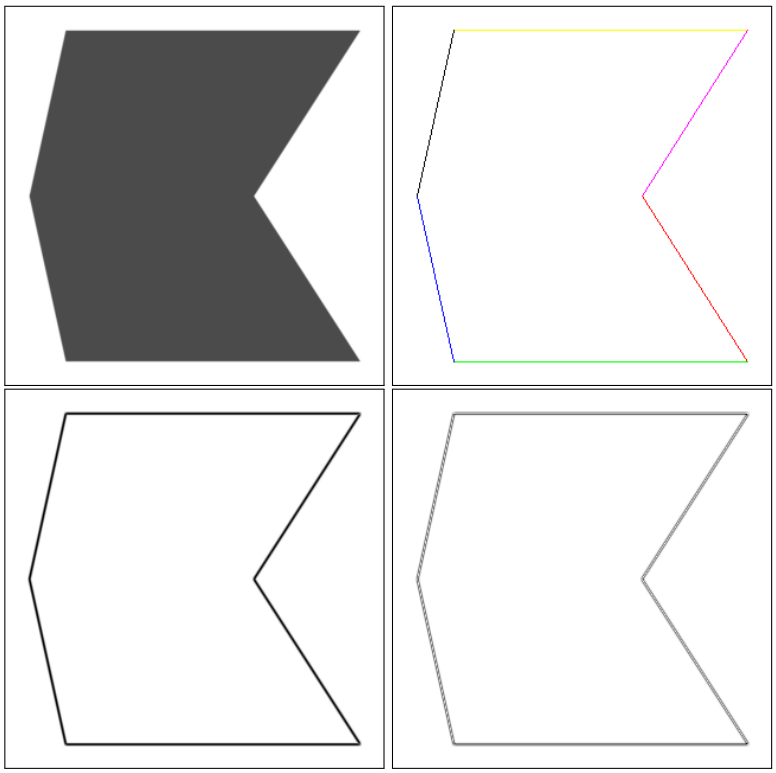


Figure 6.11: Downscaled ground truth object, ground truth line segments, Sobel 3x3 edge response and Laplace 3x3 edge response.

Edge responses were computed using a Sobel and a Laplacian filter with different kernel sizes. In the case of the first derivative responses, all non-maximum suppression variants with different interpolation options were applied. The extracted edge points were then mapped to the ground truth edges (see Figure 6.11 on the right side) to compute the distance as an error measure. In addition, the execution time for each method was determined to allow for a cost-benefit estimation. In the same way, the zero crossing methods were tested based on the second derivative responses.

There are two basic variants:

- Precise variant: In this case, the orientation is used to interpolate the orthogonal left and right magnitude values for a given point to test for a local maximum or zero crossing. An orientation map or the directional derivatives are required, even for zero crossings.
- Fast variant: Instead of interpolating the left and right magnitude values, only the directly adjacent pixels of a given point are considered, and the orientation is used to select the orthogonal left and right adjacent pixels for the local maximum test. For zero crossing, the orientation information is not needed.

Method	EP	OL	Error	StdDev	RT(ms)
threshold	1,055	3	0.3522	0.2076	6.2761
nms nearest	1,233	0	0.3132	0.1539	0.2808
zc nearest	1,053	0	0.2726	0.1231	0.2331

Table 6.2: This table compares the three main edge point extraction approaches: Threshold with morphological thinning, non-maxima suppression, and zero crossing (both at the pixel level only, denoted by "nearest", which stands for nearest neighbor). The table has 6 rows, starting with the methods. EP is the number of detected edge points after the edge extraction methods, such as nms, have been applied to the Sobel or Laplace edge responses. It shows that depending on the method, a different number of edge points are collected. OL stands for outliers. It counts all edge points that could not be uniquely assigned to an edge. The Error column shows the average distance in pixels from the detected edge points to the assigned ground truth line. Edge points that fall in regions near corners are not included. The StdDev column shows the standard deviation of the error. The last row shows the runtime in milliseconds.

The short table (see Table 6.2) gives a brief overview of the three main edge point extraction methods discussed so far. It shows the results for the thresholding (threshold estimated by the Otsu method, thinned by the morphological operator), nms, and zc methods applied to a Sobel 3x3 and Laplace 3x3 edge response computed from the ground truth image. The table also shows that the nms and zc methods are clearly superior to the thresholding method. They provide a lower error and a much better runtime: factor >20 less.

Table 6.3 shows the results for the fast nms variant of the Sobel 3x3 edge response. It returns results for all sub-variants available for nms. The first row is the main (fast) nms method, which doesn't provide sub-pixel accurate estimates. It has a quite low runtime, but the error indicates that the pixels are on average about 1/3 pixel

away from the ground truth edges, which makes sense because the gt object has non-axis-aligned edges, and even axis-aligned edges were selected so that they do not lie exactly on a pixel row in the scaled version, but between them.

Method	EP	OL	Error	StdDev	RT(ms)
nms fast spe near	1,233	0	0.3132	0.1539	0.2808
nms fast spe lin est lin	1,233	0	0.0334	0.0165	0.3028
nms fast spe lin dir est lin	1,233	0	0.0347	0.0210	0.3520
nms fast spe cub est lin	1,233	0	0.0334	0.0165	0.2958
nms fast spe cub dir est lin	1,233	0	0.0287	0.0169	0.3632
nms fast spe lin est quad	1,233	0	0.0565	0.0489	0.3061
nms fast spe lin dir est quad	1,233	0	0.0481	0.0321	0.3177
nms fast spe cub est quad	1,233	0	0.0565	0.0489	0.2948
nms fast spe cub dir est quad	1,233	1	0.0431	0.0361	0.4049
nms fast spe lin est cog	1,233	0	0.0780	0.0395	0.2945
nms fast spe lin dir est cog	1,233	0	0.0785	0.0398	0.3219
nms fast spe cub est cog	1,233	0	0.0780	0.0395	0.3031
nms fast spe cub dir est cog	1,233	0	0.0816	0.0392	0.3489
nms fast spe lin est sob	1,233	0	0.0522	0.0213	0.2967
nms fast spe lin dir est sob	1,233	0	0.0786	0.0528	0.3484
nms fast spe cub est sob	1,233	0	0.0522	0.0213	0.2921
nms fast spe cub dir est sob	1,233	0	0.0650	0.0350	0.3421

Table 6.3: Shows fast non-maxima suppression (fast nms) with all variants for pixel interpolation and estimation for a Sobel edge response with kernel size 3x3. (Sub)pixel estimation (spe) using nearest neighbor (near), in this case no subpixel optimization. The spe linear (lin) mode uses bilinear pixel interpolation, while spe cub (cubic) uses bicubic interpolation. The dir mode indicates that directional derivatives are used instead of a simple orientation map (direction is split into 4 or 8 directional regions and stored as a lookup table). The subpixel estimation method is indicated by lin (linear), quad (quadratic), cog (center of gravity), and sob (Sobel).

The method variants in all the following rows provide sub-pixel accurate estimation. They show an error reduction factor above 10, which indicates a significant gain in precision. The runtime is quite similar in all cases and so low that it is difficult to say whether there are real differences between the methods. When using a direct estimation ("dir"), there is a tendency to improve accuracy at the expense of slightly increased runtime.

It can also be observed that the variant with bicubic interpolation, directional information, and linear fitting performs best. In general, the linear and quadratic fitting methods seem to perform very well with this experimental setup. This may be related to how the ground truth was generated, so that the linear and cubic models fit best. Because of this behavior, the sob and cog cases are removed from the following tables to save some space, as they consistently perform worse.

The same is true for the "spe cub" lines. The table shows the same error results for all "spe lin" and "spe cub" variant pairs. It seems that the pixel interpolation method (bilinear and bicubic) is not relevant for sub-pixel estimation as long as no additional

Method	EP	OL	Error	StdDev	RT(ms)
nms fast spe near	1,233	0	0.3132	0.1539	0.2808
nms prec lin spe near	1,164	0	0.2934	0.1345	1.1424
nms prec cub spe near	1,128	0	0.2853	0.1289	1.2368
nms fast spe lin est lin	1,233	0	0.0334	0.0165	0.3028
nms prec lin spe lin est lin	1,164	0	0.0340	0.0167	1.1960
nms prec cub spe lin est lin	1,128	0	0.0339	0.0168	1.1918
nms fast spe lin dir est lin	1,233	0	0.0347	0.0210	0.3520
nms prec lin spe lin dir est lin	1,164	0	0.0318	0.0190	1.2208
nms prec cub spe lin dir est lin	1,128	0	0.0309	0.0183	1.2361
nms fast spe cub dir est lin	1,233	0	0.0287	0.0169	0.3632
nms prec lin spe cub dir est lin	1,164	0	0.0284	0.0175	1.5733
nms prec cub spe cub dir est lin	1,128	0	0.0287	0.0177	1.5847
nms fast spe lin est quad	1,233	0	0.0565	0.0489	0.3061
nms prec lin spe lin est quad	1,164	0	0.0544	0.0490	1.5671
nms prec cub spe lin est quad	1,128	0	0.0519	0.0472	1.5841
nms fast spe lin dir est quad	1,233	0	0.0481	0.0321	0.3177
nms prec lin spe lin dir est quad	1,164	0	0.0480	0.0325	1.6158
nms prec cub spe lin dir est quad	1,128	0	0.0471	0.0324	1.6166
nms fast spe cub dir est quad	1,233	1	0.0431	0.0361	0.4049
nms prec lin spe cub dir est quad	1,164	0	0.0365	0.0194	1.7335
nms prec cub spe cub dir est quad	1,128	0	0.0373	0.0191	1.6505

Table 6.4: This table shows a comparison of all main nms variants (fast, precise linear, and precise cubic), but excluding the cog and sob estimators and the spe cub variants.

directional data is used. In both cases, the model fitting results are the same.

Table 6.4 compares most of the sub-variants (except cog, sob, and spe cub) presented in Table 6.3 with respect to the main variants (fast, precise linear, and precise cubic). It shows that the runtime for the precise variants increases significantly by a factor of 3-5, but the precision gain, even in the base case, is marginal. This is most likely related to the small kernel size of the Sobel operator, which means that there is not much more information available that could be used to further improve the subpixel estimation.

Method		EP	OL	Error	StdDev	RT(ms)
zc near		1,053	0	0.2726	0.1231	0.2331
zc fast	spe lin	1,053	1	0.1006	0.0863	0.2382
zc fast	spe lin dir	1,053	0	0.1091	0.0535	0.2775
zc fast	spe cub dir	1,053	0	0.0966	0.0420	0.2937
zc prec lin	spe near	1,053	0	0.2726	0.1231	0.2312
zc prec lin dir	spe near	1,134	26	0.4202	0.2363	1.9374
zc prec lin	spe lin	1,053	1	0.1006	0.0863	0.2411
zc prec lin	spe lin dir	1,053	0	0.1091	0.0535	0.2705
zc prec lin dir	spe lin dir	1,134	0	0.0939	0.0470	1.9760
zc prec lin dir	spe lin	1,134	11	0.1315	0.1058	1.9539
zc prec lin	spe cub dir	1,053	0	0.0966	0.0420	0.2650
zc prec lin dir	spe cub dir	1,134	0	0.0939	0.0442	2.0398
zc prec lin dir	spe cub	1,134	11	0.1315	0.1058	1.9737
zc prec cub	spe near	1,053	0	0.2726	0.1231	0.2203
zc prec cub dir	spe near	1,143	29	0.4186	0.2372	1.8980
zc prec cub	spe lin	1,053	1	0.1006	0.0863	0.2321
zc prec cub	spe lin dir	1,053	0	0.1091	0.0535	0.2796
zc prec cub dir	spe lin dir	1,143	3	0.0936	0.0471	1.7691
zc prec cub dir	spe lin	1,143	11	0.1320	0.1065	1.7220
zc prec cub	spe cub dir	1,053	0	0.0966	0.0420	0.3030
zc prec cub dir	spe cub dir	1,143	0	0.0935	0.0443	1.9326
zc prec cub dir	spe cub	1,143	11	0.1320	0.1065	1.8664

Table 6.5: This table shows the results for the zero crossing edge point extraction methods based on a Laplacian with a 3x3 kernel. All variants with "dir" also used a direction map computed with the Sobel operator.

In Table 6.5 the zero crossing results based on a 3x3 Laplacian operator are presented. Immediately noticeable are the same error results within the variants (spe lin/cub already removed). All variants that do not use directional information for both zero crossing detection and sub-pixel estimation produce the same results within their sub-variants (compare "zc fast spe lin/cub" and "zc prec [dir] lin/cub spe lin/cub"). So the pixel interpolation method seems to be even less relevant for zc than for nms. The results can only benefit from the interpolated information if the subpixel estimation also uses directional information. Otherwise it always falls back on one of the main axes, which seems to negate the more accurate data received before.

The precision gain is only about a factor of 3. Instead of 1/3 pixel error, it is reduced to 1/10 on average. The best accuracy is achieved when using directional data in both cases. However, the results are almost the same when only the subpixel estimator is used, which significantly increases the runtime. Using the directional information for zero crossing extraction seems to provide a marginal benefit, but at a runtime cost of a factor of 7. Even faster and with almost the same precision gain are the variants without directional information. They have the great advantage of not requiring directional information, which gives a further performance boost in the case of zc.

Method	Error0	StdDev0	Error1	StdDev1
zc near	0.2300	0.0000	0.2108	0.1217
zc fast spe lin	0.1212	0.0000	0.2417	0.1530
zc fast spe lin dir	0.1212	0.0000	0.1679	0.0465
zc fast spe cub	0.1212	0.0000	0.2417	0.1530
zc fast spe cub dir	0.1212	0.0000	0.1272	0.0377
zc prec lin dir spe near	0.2300	0.0000	0.5103	0.2820
zc prec lin dir spe lin dir	0.1212	0.0000	0.1125	0.0581
zc prec lin dir spe lin	0.1212	0.0000	0.2443	0.1245
zc prec lin spe cub	0.1212	0.0000	0.2417	0.1530
zc prec lin dir spe cub dir	0.1212	0.0000	0.1116	0.0497
zc prec cub dir spe near	0.2300	0.0000	0.5034	0.2860
zc prec cub dir spe lin dir	0.1212	0.0000	0.1112	0.0588
zc prec cub dir spe lin	0.1212	0.0000	0.2434	0.1258
zc prec cub spe cub	0.1212	0.0000	0.2417	0.1530
zc prec cub dir spe cub dir	0.1212	0.0000	0.1098	0.0505

Table 6.6: This table shows some detailed information about the error estimation of extracted edges by the zero crossing variants based on a 3x3 Laplacian edge response. The mean error and standard deviation are given for the first two edges. All variants giving duplicate results are removed.

Table 6.6 shows the error estimate of the first two edge segments of the ground truth polygon. The first segment is the straight x-axis edge. It shows the average distance error for each extracted edge point to the ground truth edge. The 0 of the standard deviation indicates that all extracted points have the same distance. This makes sense since it is a straight line and can be perfectly represented by pixels. The second data pair shows the average error for an edge with a slope. In this case, the standard deviation is not 0 because the extracted edge points have different distance values. The final results are calculated based on all six edges of the ground truth object.
The last table (Table 6.7) shows an overview of the edge point extraction for several Sobel and Laplacian setups with kernel sizes ranging from 3x3 to 9x9. To reduce the size of the table, only the best subvariant is shown for each main variant (and even then some other subvariants were removed for zc since they produce the same results). The main variant without sub-pixel estimator is also included.

The main results "nms near" and "zc near" do not change with increasing filter kernel size. At least not with four-digit precision. But for the precise variants the error goes down slightly. There is also a switch of the best estimation model from "lin" to "quad" for nms as the kernel size increases. This indicates that the quadratic model is

Method	EP	OL	Error	StdDev	RT(ms)
S3 nms near	1,233	0	0.3132	0.1539	0.2808
S3 nms fast spe cub dir est lin	1,233	0	0.0287	0.0169	0.3632
S3 nms prec lin spe cub dir est lin	1,164	0	0.0284	0.0175	1.5733
S3 nms prec cub spe cub dir est lin	1,128	0	0.0287	0.0177	1.5847
L3 zc near	1,053	0	0.2726	0.1231	0.2331
L3 zc fast spe cub dir	1,053	0	0.0966	0.0420	0.2937
S5 nms near	1,227	0	0.3132	0.1539	0.3019
S5 nms fast spe cub dir est quad	1,227	0	0.0281	0.0169	0.3639
S5 nms prec lin spe cub dir est quad	1,164	0	0.0255	0.0115	1.5580
S5 nms prec cub spe cub dir est quad	1,125	0	0.0261	0.0112	1.6671
L5 zc near	1,054	0	0.2726	0.1231	0.2167
L5 zc fast spe cub dir	1,054	0	0.0370	0.0153	0.2766
S7 nms near	1,227	0	0.3132	0.1539	0.3036
S7 nms fast spe cub dir est quad	1,227	0	0.0215	0.0112	0.3548
S7 nms prec lin spe cub dir est quad	1,164	0	0.0200	0.0085	1.7692
S7 nms prec cub spe cub dir est quad	1,118	0	0.0205	0.0082	1.5858
L7 zc near	1,055	1	0.2726	0.1230	0.2377
L7 zc fast spe cub dir	1,055	1	0.0229	0.0097	0.3020
S9 nms near	1,226	0	0.3132	0.1539	0.3001
S9 nms fast spe cub dir est quad	1,226	0	0.0177	0.0085	0.3621
S9 nms prec lin spe cub dir est quad	1,162	0	0.0167	0.0069	1.5879
S9 nms prec cub spe cub dir est quad	1,114	0	0.0171	0.0066	1.6583
L9 zc near	1,055	0	0.2726	0.1230	0.2297
L9 zc fast spe cub dir	1,055	0	0.0161	0.0072	0.2941

Table 6.7: Overview of results from multiple Sobel and Laplacian setups with kernel sizes from 3x3 to 9x9. Only the best sub-variant for each main variant is shown. A threshold only setup is also included.

a better fit for "broader" edge responses. But in general, it shows the same picture as before: large precision gain as soon as the sub-pixel estimator is used, everything after that shows only nuances of improvement.

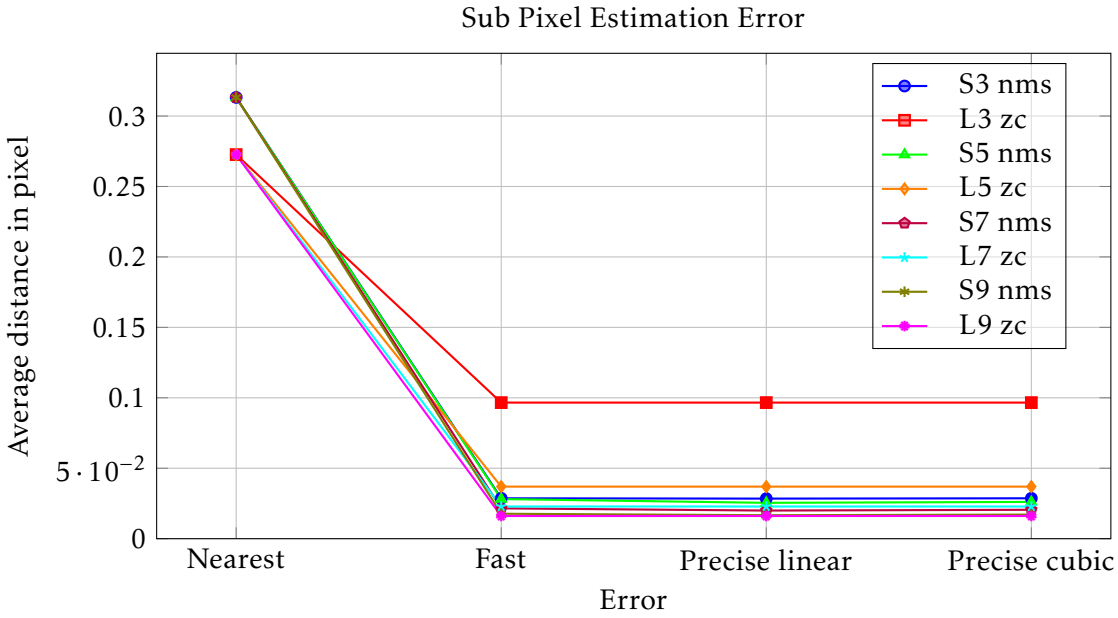


Figure 6.12: This chart shows the error results for all variants tested. For each selected category, the error is shown for the best performing variant with the lowest error.

Thus, with almost no increase in runtime (< 1.3), a large precision gain (> 10.8) can be achieved for the first nms case. While in this case there is no gain using the slower variant with directional information. An accuracy gain of up to 18.5 can be achieved by using larger filter kernel sizes (in this case 9x9). The runtime increases by a factor of > 4.3 in this case, not including the much longer processing for the edge responses.

This behavior is also visualized in the two graphs. Figure 6.12 shows how the error decreases as soon as the spe is activated, but does not really improve with even more precise calculations. The sudden increase in Figure 6.13 shows the huge increase in runtime in most cases when the more accurate computation for nms/zc and spe is enabled.

A super magnified set of line segments is plotted in Figure 6.14. It shows the lower right corner of the ground truth object with an upscale of 500. It includes lines for the ground truth itself (green line), the threshold variant (red line), nms (blue), and zc (yellow). Cyan and dark orange represent the fast spe methods for nms and zc, while dark blue and orange represent the most accurate nms and zc variants.

The orange line representing the accurate zc seems to be a bit off. While it has the best approximation to the ground truth regarding the horizontal part, the distance to the edge segment with a slope seems unexpectedly large. But since this is only a very small part of the whole segment, it could be that it is better aligned to the ground truth towards its other end. It is also possible that the reconstruction of the original polygon, defined by the corner points calculated from the intersections of the reconstructed edge segment lines, caused small numerical errors that were amplified

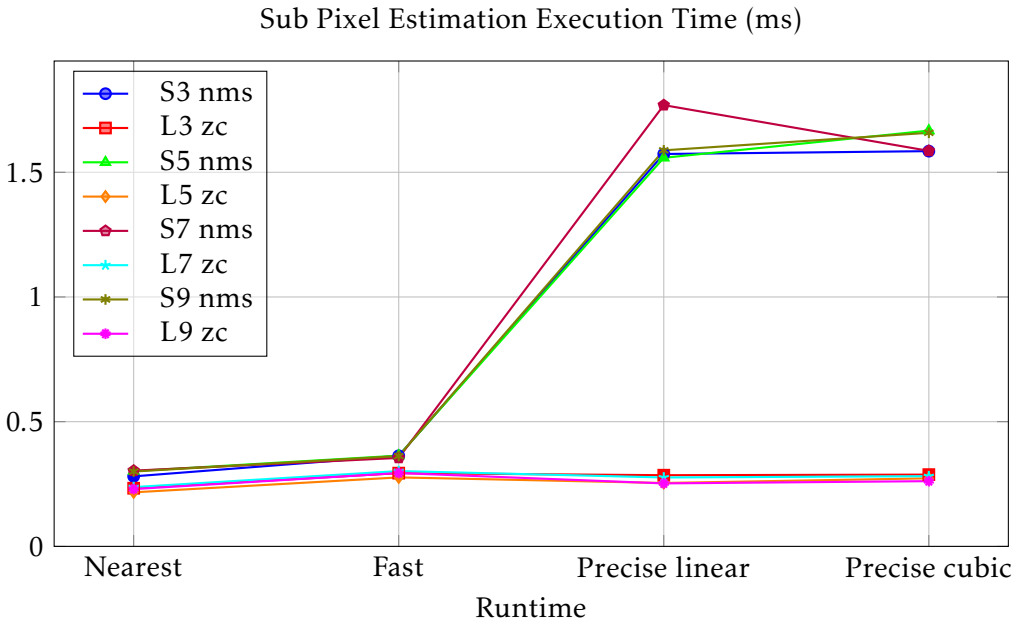


Figure 6.13: This graph shows the runtime for all variants tested. For each selected category, the runtime is shown for the best performing variant with the lowest error.

by the strong upscaling.

Remember as a takeaway: Enable the fast variant for subpixel estimation for both methods, non-maxima suppression and zero crossing, to get much more accurate edge points at almost no additional runtime cost. Everything else is only worth the extra runtime if the last bit of accuracy is really important. In this case, it also makes sense to do an analysis beforehand to find out what kind of images are involved and what kind of estimator fits best.

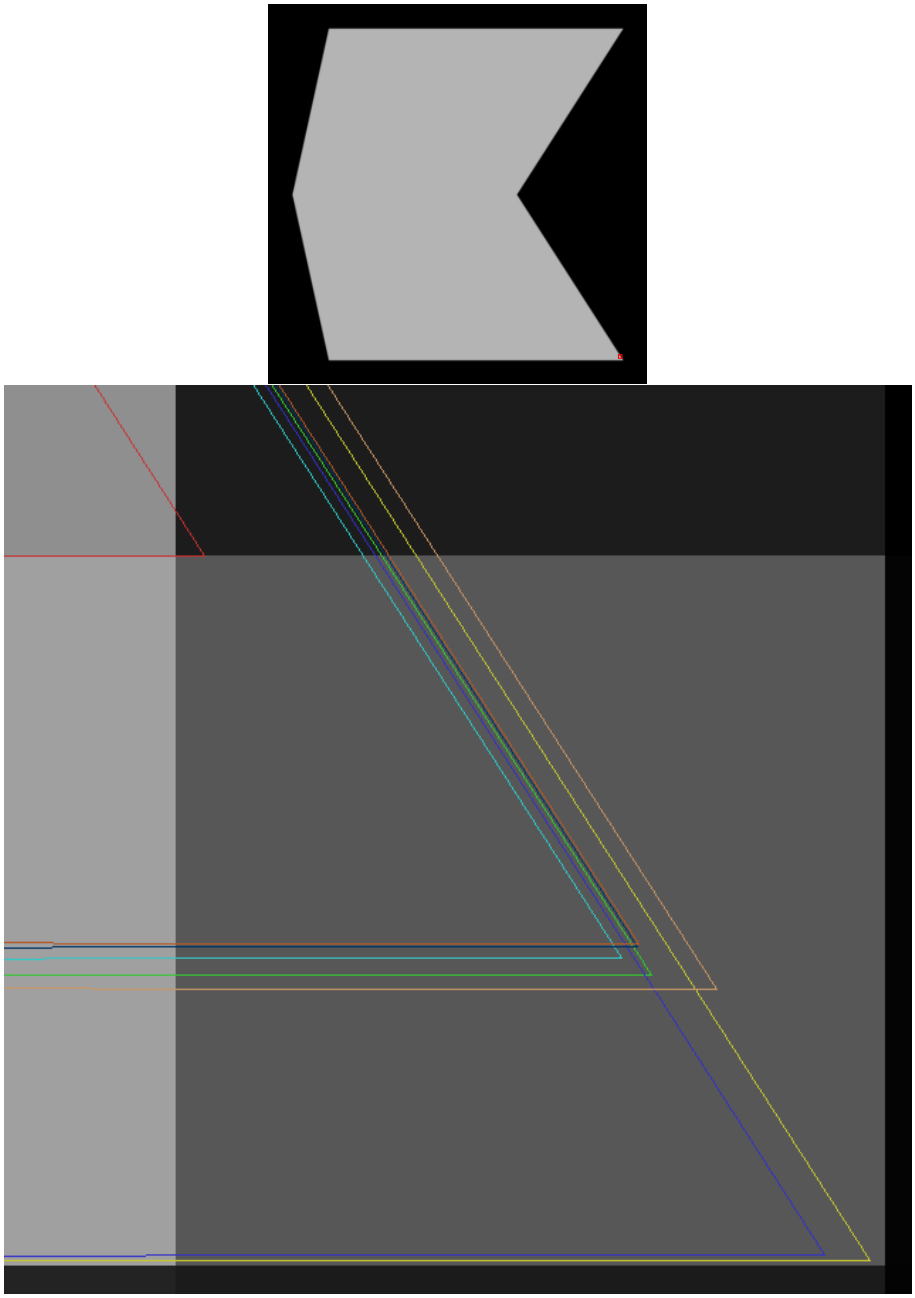


FIGURE 6.14: This figure shows the upscaled polygon lines from the ground truth and the reconstructed polygon used to generate the ground truth image. It shows, as indicated in the upper image, the lower right corner of the ground truth object upscaled by a factor of 500. Color coding: gt as green, threshold as red, nms as blue, zc as yellow, nms fast spe as cyan, zc fast spe as dark orange, S9 nms prec lin spe cub dir est quad as dark blue, and L9 zc fast spe cub dir as orange.

Chapter 7

Connected Edge Components

With the hysteresis step in the edge detection process, Canny proposed a novel approach to extracting the edge pixels by avoiding the streaking effect. While this method greatly improves the resulting edge maps, the local dependency of each pixel was sacrificed for a more global relationship. The computational cost is linear and not very high, but the global dependencies make this approach very difficult to parallelize for more efficiency in real-time applications [138]. In addition, in the original implementation, the resulting edge map discards the connected component information in the final result. While this can help optimize the algorithm for efficiency and is not necessarily relevant for post-processing (e.g., using the Hough transform to extract lines), important information is lost.

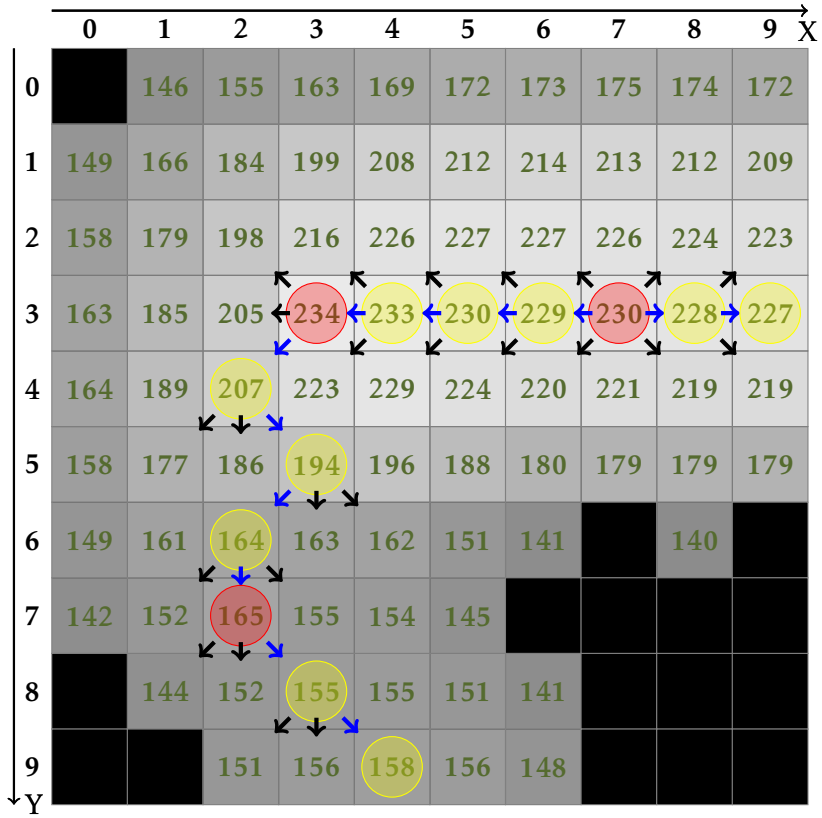
However, many post-processing approaches can benefit from this additional information. For example, when extracting line or curve segments. The information can be expressed as ordered lists of edge pixels, defining edge segments with connected edge pixels.

7.1 Edge Drawing

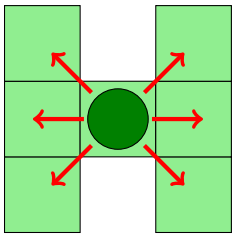
In the literature, Topal et al. [211, 210] have adopted this idea for their edge extraction method called Edge Drawing. Instead of extracting all edge pixels with non-maximum suppression, they select only a subset of them as seed or anchor points. This is achieved by searching only for maxima on a grid (e.g., every 5^th row and 5^th column). Other approaches to finding seed points are also possible, e.g. using a corner detector. However, the more sparse the seeds are, the more likely it is to miss important information in the image. The authors suggest choosing the seed size depending on the smoothing applied to the image for noise reduction. To be sure that no information is lost, a full NMS must be performed and used as seed points.

After extracting the seed points, they are used to start a smart routing method that walks on the gradient peaks to find the connected edge segment, as shown in Figure 7.1. Depending on the gradient direction, the ridge walking process compares the three adjacent edge energies of a pixel for the highest value and jumps to the highest adjacent pixel to continue the search. The search stops if it moves out of an edge response region or if it encounters an edge pixel that has already been detected. The authors' implementation uses only two directions (horizontal and vertical) because the authors found that more directions don't give better results, but are more expensive to compute.

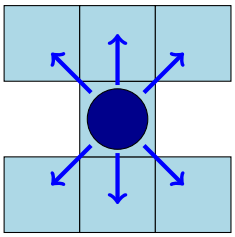
This approach produces well-defined, ordered, single-pixel-thick edge segments. Compared to NMS, it also has less problems with gaps at junctions and no problems



(A) Example edge response map.



(B) Edge drawing with horizontal search direction.



(C) Edge drawing with vertical search direction.

FIGURE 7.1: This is how the edge drawing algorithm of [211] works. (a) Shows an edge map with an example edge drawing path. It starts at a seed point (local maxima, marked with a red circle, in this case the maxima at 230) and follows the edge response in the direction of the gradient (see the red and yellow circular markers). The arrows show the options depending on the horizontal (b) or vertical (c) search rule. The blue arrow indicates the selected option. From the starting seed, the edge is drawn to the left and to the right.

with gaps at poorly localized edge regions. It is also possible to compute sub-pixel positions to increase accuracy, similar to the NMS method.

A major drawback of this approach is that some edge pixels are not at the true maxima. Also, the detection of junctions is not taken into account, leading to incorrectly connected segments. To guarantee that all edges are extracted, the full NMS must be applied as preprocessing, making the computational improvement by computing sparse seeds obsolete.

A simple rule to avoid false fragmentation at nodes is to test for strong edge response changes, which typically occur when a segment is occluded by another segment. In this case, the connectivity of the foreground edge is favored, which is a desirable effect.

7.2 Edge Linking

To overcome the problems of NMS and the drawbacks of Edge Drawing, a more sophisticated method is proposed in this section. When Edge Drawing is combined with a Canny hysteresis and NMS in a more convenient way, the processing can be even faster and more accurate than Edge Drawing (with full NMS) by using all the information extracted by the NMS processing (see Figure 7.2 for examples). It is also possible to extract more information in the connected component process with minimal additional cost. By marking endpoints, corners, and junctions as nodes and the connected components as edges, a feature graph can be easily constructed.

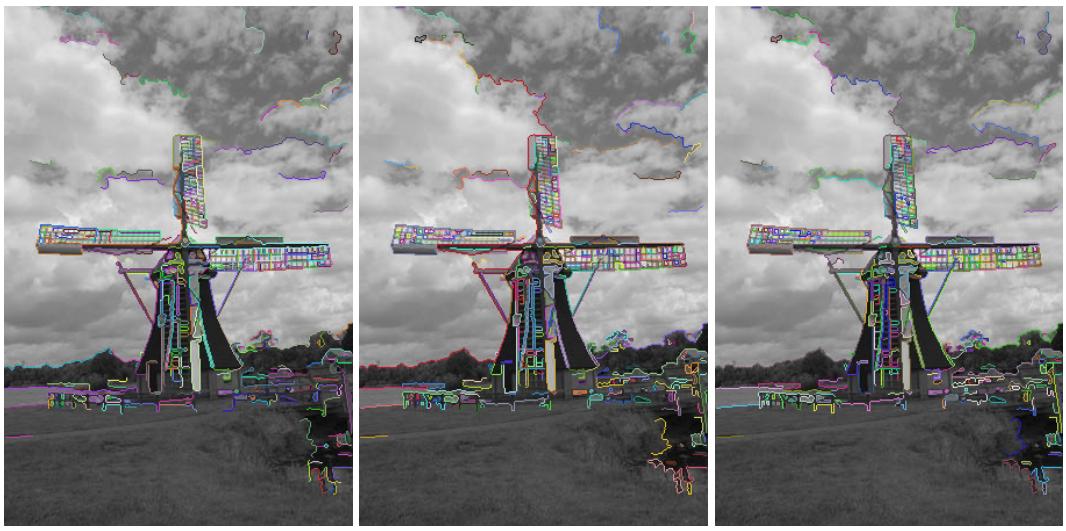


Figure 7.2: Examples for edge methods. Left: Edge Drawing. Center: Edge Linking, Right: Edge Pattern Linking. The images show the results of the non-maximum suppression (NMS) step processed with Otsu estimated thresholds. The coloring is chosen at random to distinguish the segments.

7.2.1 Basic Algorithm

The core idea of this approach is to apply a variant of edge drawing to the edge pixels extracted by NMS. During the NMS process, edge directions are computed (for fast NMS, only the four quadrants are considered, which is still sufficient). Instead

of discarding this information, it is stored in a map. Additionally, seed points are generated during NMS and stored in a separate list. These points are added when the edge response exceeds the higher threshold. All maxima exceeding the lower threshold are recorded in a 2D edge map.

The basic and fast connected component process consists of the following steps:

1. Start at a seed point and perform a forward and backward search.
2. Search for a connected edge point by predicting the adjacent point in the direction of the edge orientation of the seed or current point. If the predicted pixel is a match in the NMS edge map, the search continues from there.
3. If the predicted pixel is not in the NMS map, try to find an NMS point one pixel to the left or one pixel to the right of the predicted point.
4. If no point is found, start over with the next seed point.

This approach outperforms edge drawing in computational efficiency because the prediction step requires fewer operations than edge energy comparison. However, several artifacts may be present, such as those caused by noise or poorly chosen thresholds for NMS. The problem of gaps in the NMS map can be addressed by incorporating a gap detection into the connected component search process. A practical method is to analyze the edge response when no maximum is present and use this information to continue the segment. The Edge Drawing method is particularly well suited for this task and can be integrated as a gap compensation step after step 3 of the basic algorithm:

3a. If no NMS pixel was found, use Edge Drawing to find the highest edge response value. If this value is above the lower threshold, continue with that pixel.

With this modification, a hybrid approach has been developed that retains the benefits of hysteresis, NMS, and Edge Drawing while remaining more efficient than original Edge Drawing alone. However, one problem that can still occur with NMS is the presence of thick edge segments that are two pixels wide. This can result in either disrupted connected edge segments that do not consistently maintain a single-pixel thickness, or multiple separate edge segments representing the same edge. If it is critical for post-processing that the resulting edge segments meet these conditions, a simple rule can be implemented to remove the problematic NMS pixels:

3b. If an adjacent pixel is found, remove NMS pixels adjacent to the current pixel that are orthogonal to the move direction of the found pixel.

This can result in unwanted gaps, which are easily handled by gap detection. Step 3 still doesn't consider what happens if two adjacent NMS pixels are found. A simple rule would be:

3c[1]. If two adjacent NMS pixels are found, use the one with the greater edge response to continue.

However, this information also indicates that there is likely to be a transition at this location. If the segment is simply continued, there is a high probability of creating a

false edge segment by connecting two unrelated edges. A simple solution is to split the segments at junctions. The basic algorithm can be further extended:

3c[2]. If two adjacent NMS pixels are found, stop the current segment and use the adjacent pixel as a seed for a new segment.

Another breaking rule (corner rule) can be added to detect hard corners in the segmentation process:

3d. If the next two or three found edge pixels indicate a change in direction greater than 45°, stop the segment at that point and continue a new segment at the next point (see Figure 7.4 for details and Figure 7.3 for a concrete example).

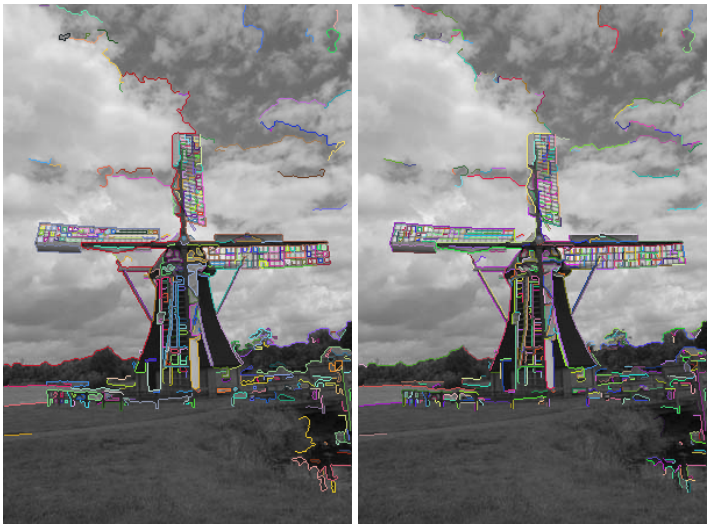


FIGURE 7.3: Examples for the edge linking method. On the left side the basic variant is shown while on the right side the corner rule is activated.

Since the direction check has to be performed for each step of the search, it is quite expensive. It can be done more efficiently with some post-processing, e.g. by decomposing the edge segments into linear line segments as described in chapter 8.

A later process can reconnect related edge segments if they were mistakenly disconnected. To find the related edge segments, a graph created in the process of tracing the edges can be used. To decide if they need to be reconnected, the curvature and continuity of two segments at their transitions can be analyzed.

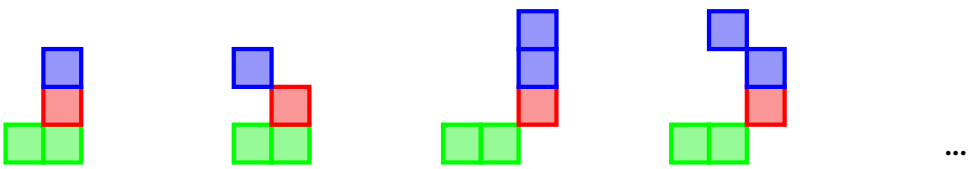


FIGURE 7.4: Examples of corner split rules. The red color indicates the current pixel in the segmentation process, the green color indicates already visited pixels, and the blue color indicates upcoming pixels.

Other information that can indicate segment connections or breaks are edge orientation changes, e.g. when the gradient is reversed by 180° because the background changes at an edge. Until now, this information has been ignored by simply considering only the range $[0, \pi]$ for the direction of an edge. This can be easily changed by slightly adjusting the NMS process to use the whole range of 2π.

This algorithm has been used in the line segment-based extension of ELAS (Efficient Large Scale Stereo Matching) [63], known as LS-ELAS [1]. Rather than generating support points using a uniformly sampled grid of the image, LS-ELAS uses line segments to determine these points. This approach allows the identification of highly informative support points that effectively preserve depth discontinuities, improving the overall performance of the stereo matching algorithm.

7.2.2 Graph Extension

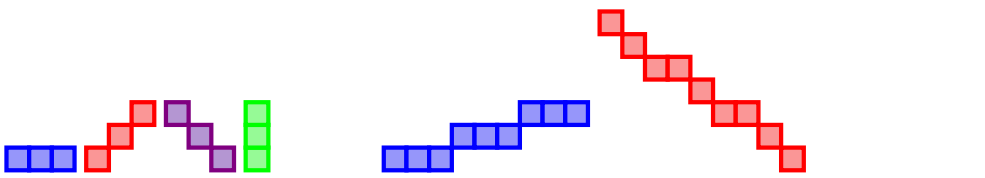
A directed graph $G = (N, E)$ can be used to store the relations between edge segments, e.g. segments connected by corners and junctions. The nodes $n_i \in N$ express the segment endpoints, corners and junctions (the type is also stored), and the edges $e_j \in E$ represent the directed edge segments. This information can be used to efficiently find adjacent or related edge segments, e.g., to merge segments of similar orientation to create large line segments, to infer occlusion hypotheses, or for more stable feature matching.

7.3 Edge Pattern Linking

Until now, the edge segment extraction problem has been considered as a simple connected component approach based on single edge pixels. In this approach, an intermediate step is introduced that divides the connected component processing into two steps:

- Find patterns with consistent primitives of connected edge pixels
- Find connected patterns as final edge segments

The same algorithm for finding connected edge pixels introduced in section 7.2 is used to find patterns based on linear primitives [117]. Additional constraints are added to force a new pattern when the starting primitive of the current pattern is not repeated. An additional parameter allows to control some variance within the primitives, allowing to create patterns with similar repeated primitives.



(A) Linear Primitives (B) Simple Patterns ...

FIGURE 7.5: Example of primitives on the left and patterns composed of primitives on the right.

The basic idea of this method was to reduce the effort of finding straight edge segments. Straight edge segments can be represented by a single pattern or a few alternating patterns. Allowing some variance in the primitives can further reduce the number of patterns representing a straight edge segment (see Figure 7.5). This approach has some additional advantages: corners can be found more easily and efficiently, and assumptions about the curvature of an edge segment can be derived from the number of patterns within a line segment. A more robust and efficient corner rule can be implemented, providing better results for line segment extraction in combination with the split and merge method as discussed in chapter 8.

As shown in the analysis part of chapter 8, the computational overhead for the pattern constraint is slightly greater than the computational savings in the line extraction process. However, the additional information provided and the greater control over line segment extraction make this approach superior to the previous method.

7.4 False Edge Segment Detection

Up to this point, false edge responses have been addressed by introducing a threshold to eliminate weak responses. However, as discussed in subsection 6.1.4, determining an optimal threshold that effectively filters out noise while preserving true edges is a challenging task. An alternative approach is to use very low thresholds that remove only irrelevant responses (e.g., a single intensity change that produces the smallest possible response) while allowing more noise to pass.

Noise detection can be managed more effectively by considering structural properties, in this case, the connectivity of edge response pixels. Because noise typically produces local peaks without forming meaningful structures, it often results in small, isolated edge segments. A simple but effective method for removing false segments caused by noise is to apply a threshold t_p to the number of supporting pixels N_{sp} within an edge segment.

- A segment is rejected if $N_{sp} < t_p$ and if it is not connected to another segment that has enough supporting pixels.

However, it is also possible for larger edge segments to be caused by noise, especially at locations where some signal is present or nearby, or for segments caused by noise to be attached to good segments. To detect such segments, a segment validation method is required that can distinguish between meaningful and meaningless segments.

Desolneux et al. [40] proposed a statistical edge detection method based on computational Gestalt theory and the Helmholtz principle [41]. According to their approach, a geometric structure (in terms of grouping or Gestalt) is considered perceptually meaningful if its occurrence would be extremely unlikely in a purely random scenario. This statistical significance is quantified by the *Number of False Alarms (NFA)*, which estimates the expected number of similar detections that would occur under a noise-only model. A low NFA value (typically < 1) indicates that the detection is highly unlikely to be a random fluctuation and should therefore be considered meaningful.

This concept follows an "a contrario" approach, where geometric structures are detected as outliers within a noise model. For this method to be valid, the noise model must assume that the intensity values of each pixel in an image are independent.

Consequently, this also implies that both the direction and magnitude of the image gradient are independent.

As a simple noise model that meets this requirement, Desolneux et al. used Gaussian white noise. Based on this model, they analyzed all the level lines (in this case lines, contours or shapes with same intensity: $\{(x,y)|I(x,y) = c\}$, also see subsection 8.2.2 for more details) in an image and identified meaningful level lines by comparing the gradient distribution of each level line segment to the overall gradient distribution across the entire image. However, this results in a computationally expensive algorithm that produces poorly localized and over-detected edges.

Nevertheless, their validation process can still be useful for assessing whether previously detected edge segments are meaningful. Akinlar and Topal [3] built on this idea by incorporating the validation step, including the NFA-based significance test, into their edge drawing algorithm, thereby reducing its dependence on edge response thresholds.

Before examining the final validation method proposed by Akinlar et al., the basic edge segment validation approach of Desolneux et al. is first introduced.

7.4.1 Contrast based Segment Validation

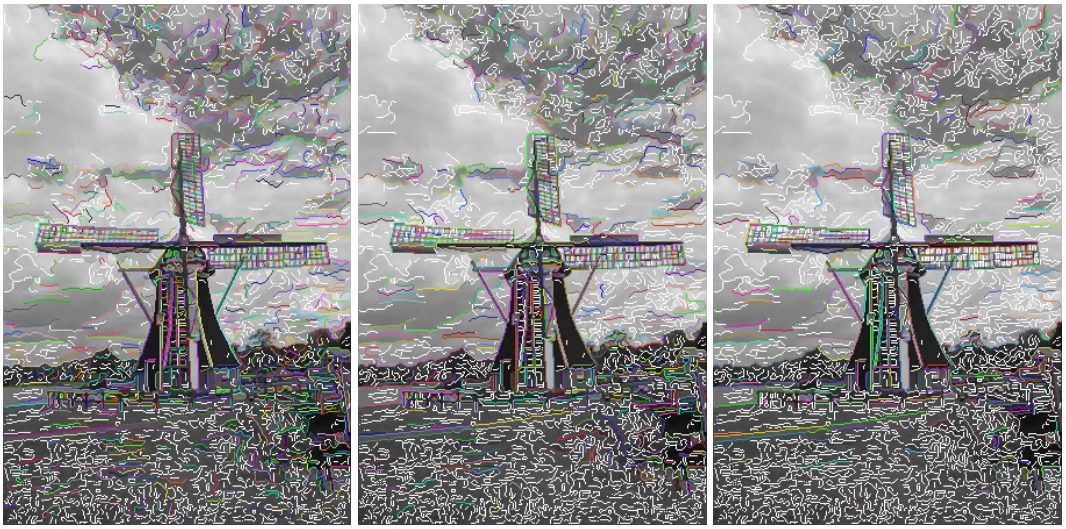


Figure 7.6: Examples of contrast-based NFA applied to the Edge Linking method. On the left, a large NFA of 100 is selected. In the middle is the default case with an NFA of 1. On the right, an NFA of 0.001 was used. The colored edge segments passed the NFA test, all others (white) failed.

Let I be a discrete grayscale image with size MxN and intensity range $[0,r]$ (by default $r = 255$). Let E be the edge response, e.g. by computing the norm of the first derivative (gradient) of the image I. Let H be the empirical cumulative distribution of the edge response E as a function of the given threshold μ, and C the number of image pixels with $E \neq 0$:

$$H(\mu) = \frac{1}{C} \sum_{\forall x \in I} \begin{cases} 1 \; if \; E(x) \geq \mu \\ 0 \qquad\qquad else \end{cases} \tag{7.1}$$

In order to calculate the NFA, it is necessary to know the number of level lines and all sub-lines (segments or subdivision of a level line up to a minimum unit of 2 pixels). This can be done by first computing the number of sub-lines $SL = \frac{li(li-1)}{2}$ for each level line L_i with length l_i (number of pixels of L_i) and then accumulating the sub-lines to the final number of lines:

$$N_l = \sum_i \frac{li(li-1)}{2} \tag{7.2}$$

The NFA of a level line or sub-line can now be estimated by:

$$NFA(L) = N_l H(\mu)^l \tag{7.3}$$

The variable l represents the length of the level line L (number of pixels belonging to the level line). The variable μ is defined as the minimum edge response of all pixels $x_1, x_2, \ldots, x_l$ of L. For the level line L, NFA calculates the number of false alarms under a given noise model (in this case white noise). Then L can be called ε-meaningful if $NFA(L) \leq \varepsilon$. Desolneux et al. suggest choosing $\varepsilon = 1$, which means one false alarm for a given geometric structure of the image.

This NFA function can also be applied to edge segments. In this case, only the number of level lines and sub-lines needs to be adjusted by the number of edge segments and sub-segments. To evaluate a level line or edge segment, all subparts must be evaluated. They will be discarded if the evaluation fails:

1. For each sub-line of L test if $NFA(SL) \leq \varepsilon$.
2. Return L or the valid parts of L

Testing all parts of a line or segment is computationally expensive. Derived from the properties of NFA defined by Desolneux et al., Akinlar et al. state the following:

1. For two segments with the same minimum edge response μ, the longer one is more significant.
2. If a line L of length l with minimum edge response μ is invalid, then no sub-line of L containing the pixel with minimum contrast can be valid.

These statements can be used to implement a more efficient recursive validation algorithm:

1. Test $NFA(L) \leq 1$, if true, return true
2. Split L into two sub-lines at the pixel with the minimum edge response and also skip that pixel for sub-lines.
3. Validate both sub-lines by starting at 1.

This algorithm can validate an edge segment in linear time. See Figure 7.6 for an example.

7.4.2 Orientation based Segment Validation

In the later work of Desolneux et al. [41] they stated in Principle 2 that the interpretation of images should not depend on the intensity values, but on their relative values (based on Wertheimer's contrast invariance principle). To be independent of contrast information, the gradient orientation would be much more reliable than the gradient norm. When considering the latter method, the dependence on image intensity can lead to the failure to detect meaningful segments. For example, a synthetic binary image with a black square on a white background will produce a gradient with zeros everywhere except the square boundaries. The gradient histogram has only one count N for a single response with intensity R. So $C = \#E \neq 0 = N$, $\mu = R$, $\#E \geq \mu = N$ and $H(\mu) = \frac{N}{C} = 1$, which means that no segment is meaningful. The problem can be solved by adding some noise or applying a slight blur. For natural images, this case can practically never happen.

For lines, the orientation alignment of the support points can be used to define an NFA test. In this case, Desolneux et al. used a binomial distribution as follows

$$NFA(l,k,p) = N_s \sum_{j=k}^{l} \left(\begin{array}{c} l \\ j \end{array} \right) p^j (1-p)^{l-j} \tag{7.4}$$

where N_s is the number of possible line segments in a NxM image. Since line segments have two endpoints that can be anywhere in the NxM pixels, it is $N_s = (MN)^2$. The probability p defines the accuracy of the line direction. Depending on p, the number of aligned sample points k is determined, while l is the length of a line segment S (as number of contained pixels). This method has been adapted by Gioi et al. to validate rectangular line support regions [67] and by Akinlar et al. to test for meaningful line segments [2]. As angle tolerance t both use $\frac{\pi}{8}$ by default, so $p = \frac{1}{8}$ since $p = \frac{t}{\pi}$, $0 \leq p \leq 1$.

To apply this approach to edge segments, only the orientation alignment needs to be adjusted. The number of potential segments remains the same because an edge segment also has two arbitrarily located endpoints. An edge segment does not necessarily have an orientation. To estimate the number of well aligned pixels within the edge segments, the absolute difference of the orientation of two adjacent support pixels is calculated. If the difference is less than a threshold d, then the number k of aligned pixels is incremented. Again, a threshold of $\frac{\pi}{8}$ is used. This allows smooth curved objects to be aligned well. The length of the segment is determined by the number of support points minus one, because no more good aligned orientations can be obtained with this method. Again, ε is set to 1, accepting on average one false detection per image by the given noise model.

The final validation algorithm is similar to the algorithm of Akinlar et al. Instead of searching for the minimum edge response, the maximum orientation difference within the given segment has to be found. The segment is split at this point if the test fails. Another similar solution would be to first split the segment into line segments (see subsection 8.3.2) and then apply the NFA test. Since no expensive repartitioning function $H(\mu)$ needs to be precomputed, the validation process is much faster in both cases than in the first approach. For an example see Figure 7.7.

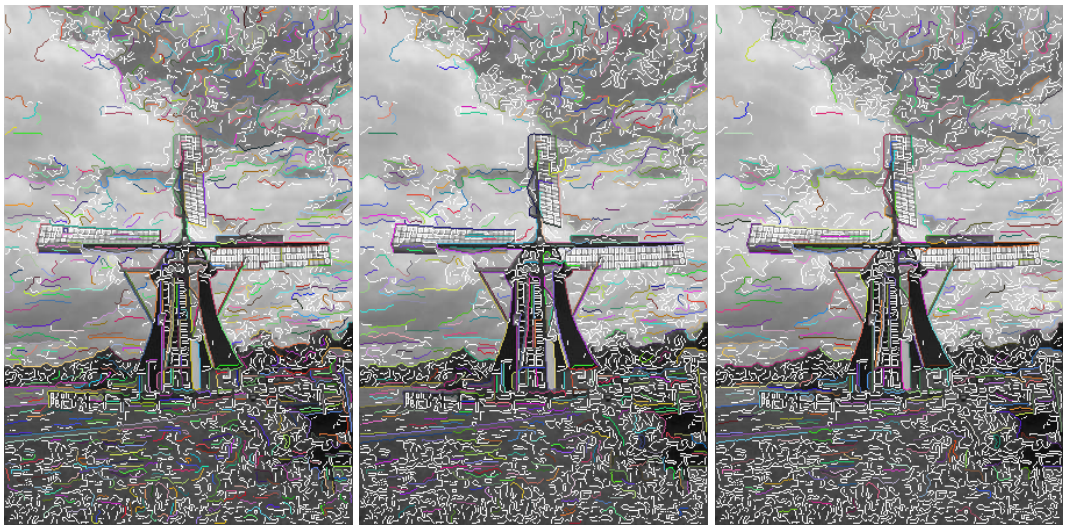


FIGURE 7.7: Examples of orientation-based NFA applied to the Edge Linking method. On the left, a large NFA of 100 is selected. In the middle is the default case with an NFA of 1. On the right, an NFA of 0.001 was used. The colored edge segments passed the NFA test, all others (white) failed.

7.4.3 Further Refinements

Gioi et al. suggested using $-log10(NFA)$ because it is more intuitive than NFA:

- **...**
- -2: corresponds to 100 average false alarms
- **-1**: corresponds to 10 average false alarms
- **0**: corresponds to one average false alarm
- **1**: corresponds to 0.1 average false alarms
- 2: corresponds to 0.01 average false alarms
- etc.

The larger the value, the more meaningful the segment. The logarithmic scale brings the results closer together and gives more influence for ε, since before really significant changes for ε were needed to see any effect. This change is used for all NFA validation methods.

The value returned by the NFA test is not only useful for rejecting meaningless segments. It can also be used to find the most meaningful elements from a set. This is a convenient way to reduce the number of features when too many have been detected, e.g., by selecting the 200 most meaningful segments (see Figure 7.8).

7.5 Results and Discussion

In this chapter, two main topics were introduced, which will now be examined in more detail with regard to their runtime behavior and usefulness for line segment extraction.

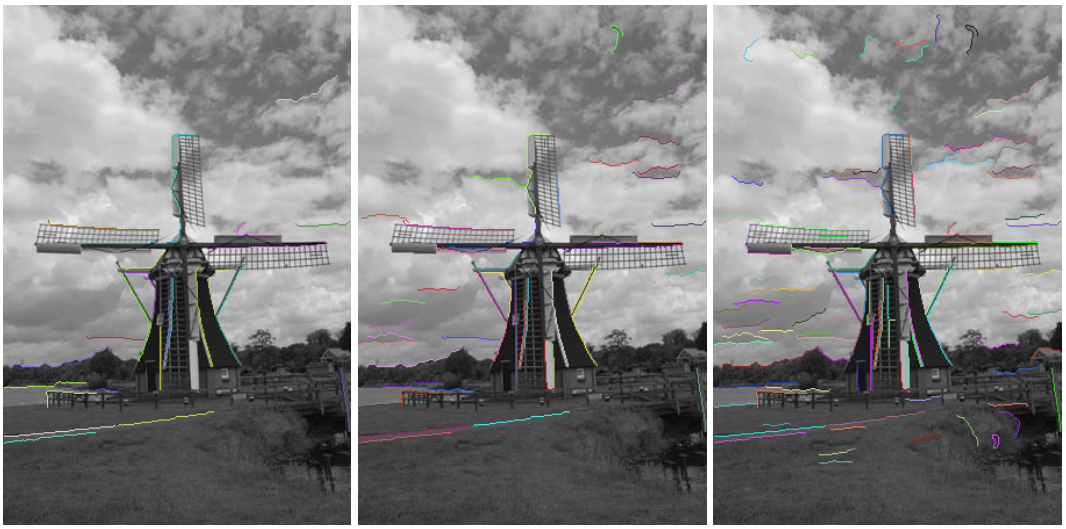


FIGURE 7.8: Examples of orientation-based NFA applied to the Edge Linking method and sorted by the most meaningful segments. From left to right: Top 25, Top 50, and Top 100 most significant edge segments.

7.5.1 Edge Segment Detection (ESD)

For this thesis, four variants of the edge segment detection (ESD) algorithm were implemented and evaluated. The first variant (ESD Simple) is a simple connected component approach based on the basic algorithm described in section 7.2. The second variant (ESD Drawing) is based on the original Edge drawing approach (as proposed of Topal et al.), using gradient peaks to walk along the edge segments instead of predicting the next pixel from the NMS map. The third variant (ESD Linking) is a more advanced version of the basic algorithm that incorporates gap compensation and corner detection. The fourth variant (ESD Pattern) is a pattern-based approach, as described in section 7.3, that uses linear primitives to find edge segments.

All variants were tested on the same sets of images, which were selected to cover a wide range of edge types. The images were preprocessed by a Sobel derivative filter (3x3 kernel, see section 4.1) to get the edge responses and the edge points were extracted by a fast nms method using a 8 zone edge direction map (see subsection 6.2.2). Low values for the nms thresholds were selected (lower threshold = 0.004, upper threshold = 0.012), to get a realistic performance (many edge points are present including noise and low contrast edges).

The tests were performed on a high performance HP ZBook 15 Fury G8 notebook with 32GB RAM, Intel Core I7 CPU (11th Gen i7-11850H @ 2.50GHz, 8 Cores) and NVIDIA RTX A3000 Laptop GPU, running ubuntu 22.04 (jammy). As in previous experiments (see section 5.1), the algorithms were executed based on the datasets BSDS500 and MDB-X.

Table 7.1 gives an overview of the runtime of the 4 ESD variants. Additionally a runtime for the ESD Linking and ESD Pattern variants with activated corner rule (CR) are listed. As mentioned earlier, the ESD Simple variant performs better then the original Edge Drawing method. But the combined a bit more robust method ESD

Method	BSDS500	MDB-Q	MDB-H	MDB-F
ESD Simple	0.943	1.499	5.094	18.842
ESD Drawing	1.299	1.919	6.567	23.901
ESD Linking	1.334	2.016	7.146	28.996
ESD Pattern	1.680	2.592	9.410	35.626
ESD Linking CR	1.791	2.655	9.577	36.290
ESD Pattern CR	1.759	2.676	9.700	37.684

Table 7.1: Runtime for the ESD variants in milliseconds. CR stands for Corner Rule.

Linking performs similar to the ESD Drawing method. This is due to the fact that more rules are applied that cancel out the runtime advantage if the simple variant.

The pattern variants are the slowest. Although they actually summarize more information and are therefore in some ways more efficient to handle, the effort required to extract this information is greater than with the simple edge segment approaches.

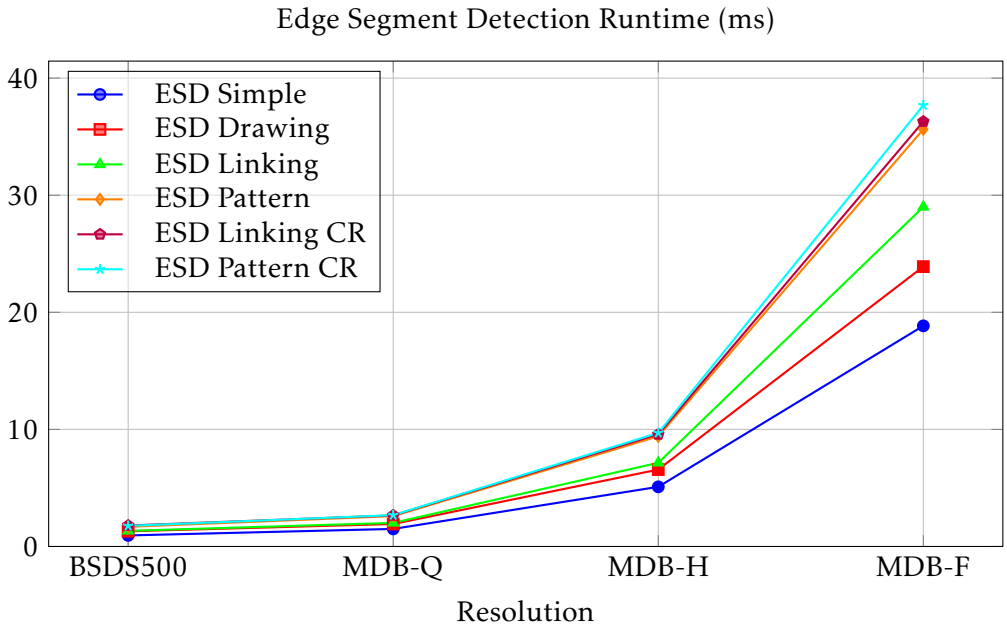


Figure 7.9: This chart shows the runtime behavior for the ESD methods in milliseconds. CR stands for Corner Rule.

Figure 7.9 shows the runtime of the ESD algorithms in a chart related to the image size. As expected, the runtime is quadratic. This means that the ESD algorithm itself has a linear runtime, as the quadratic component is already caused by the quadratic increase in pixels.

What stands out in the diagram in terms of runtime are the pattern variants. They are separated from the rest with a much higher runtime, with one exception. But both are very close to each other. This means that further options such as the corner rule have little effect on the runtime due to the already increased complexity, while the

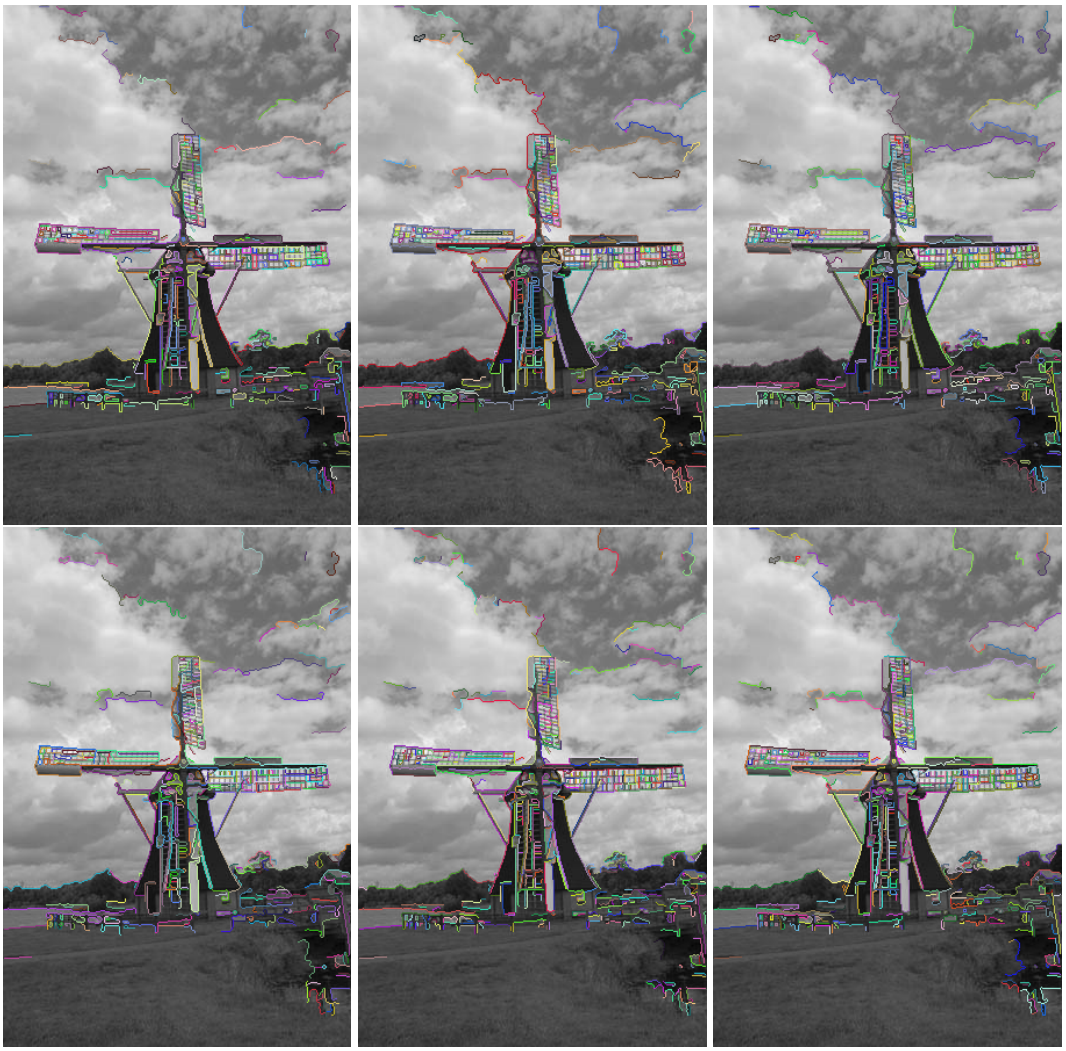


FIGURE 7.10: Examples for all edge methods. Top left to right: Edge Simple, Edge Linking and Edge Pattern Linking. Bottom left to right: Edge Drawing, Edge Linking with corner rule, Edge Pattern Linking with corner rule. The images show the results of the non-maximum suppression (NMS) step processed with Otsu estimated thresholds. The coloring is chosen at random to distinguish the segments.

corner rule significantly increases the runtime for the faster linking method (compare ESD Linking CR, ESD Pattern, and ESD Pattern CR).

In summary, the simplest and fastest option is usually a good choice (also compare the visual results in Figure 7.10. There are hardly any differences). Many problems will be solved in the further course of line segment processing (see next chapter). When it comes to extracting connected edge segments that are as long as possible, changing the method can certainly provide better results.

7.5.2 Meaningful Edge Segments

This section compares the three available NFA implementations. As described in section 7.4, there is one NFA test implementation using edge response strength (contrast based) and a second and third implementation using edge orientation. The second variant is based on the original implementation by Gioi et al., while the third variant is a custom implementation optimized for edge segments.

To compare the three variants in terms of runtime, the same datasets and settings were used as for the ESD tests. The hardware setup is also the same.

Method	BSDS500	MDB-Q	MDB-H	MDB-F
NFA Contrast	15.335	27.580	92.774	322.071
NFA Binom	0.951	1.407	5.346	20.092
NFA Binom2	1.024	1.444	5.214	20.088

Table 7.2: Runtime for the NFA variants in milliseconds.

Table 7.2 shows the runtime values for the three NFA methods. As mentioned before, the contrast variant is computationally expensive, which corresponds to the slowest runtime in the table. The other two variants are much faster, with similar runtimes. This is to be expected, since both implementations are based on the same approach.

In the context of line segment extraction, the question arises whether it is worth to integrate the NFA test directly into the process. Although the NFA test provides efficient ways to evaluate the significance of segments, it is often found in practice that the selection of a well-estimated threshold can provide similar results without the additional runtime overhead of the NFA test.

A comparison between an Otsu threshold and the combination of a low threshold and NFA shows that an appropriate threshold is often the more robust and resource-saving solution (see Figure 7.11). It is therefore advisable not to integrate the NFA test permanently into the extraction process, but to use it independently and subsequently if an additional evaluation or ranking of the extracted segments is required.

An example could be an optimized line feature matching approach (see section 9.3), where only a limited number of edges can be processed. By subsequently applying the NFA test, a selection of the "best" and most reliable edge segments can be made, improving the quality of the matching results. In this scenario, the NFA test serves as an effective method for weighting and prioritizing already extracted segments without unnecessarily slowing down the extraction process itself.

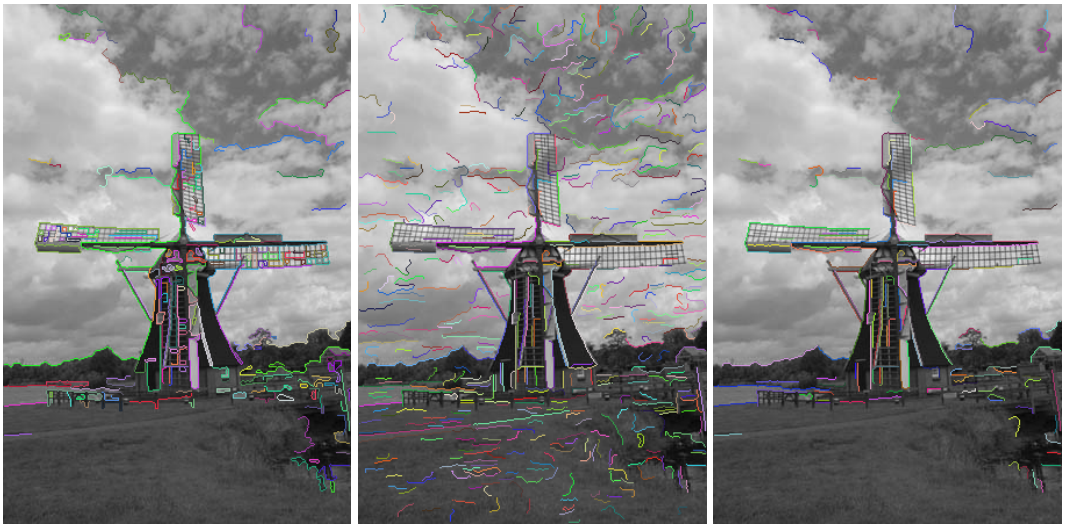


FIGURE 7.11: Examples for Edge Linking with proper (Otsu) threshold on the left, Edge Linking with NFA 1 (1 average false alarm selected) in the center and Edge Linking with NFA 1 and Otsu threshold on the right. In this case the Edge Linking method also used a 10 pixel threshold to remove "unimportant" segments. This filter was not applied in the NFA cases.

Chapter 8

Line Segment Extraction

In the last decades, many line extraction methods have been developed. One of the most popular methods is the Hough transform. It uses an abstract approach by transferring each edge pixel into a certain parameter space (also called Hough space). Other methods use a more direct approach by working with the edge pixels in image space. They use variants of connected components algorithms to find coherent edge pixels as line support regions. These line support areas are then used to fit straight lines or curves.

All of these methods have a common initial processing step: computing the edge pixels. This leads to the following common initial steps

1. Image preprocessing to reduce noise (see chapter 4 and section 3.3)
2. Edge response estimation (see chapter 4)
3. Extraction of edge pixels (see chapter 6)

Some advanced methods may combine or skip parts of the initial steps. For example, the RCMG operator can skip step 1 because step 2 is optimized to be robust against noise. Many gradient operators, such as the Sobel operator, already include smoothing and can also skip the first step for images with good quality and high SNR. Depending on the line segment extraction method, step 3 can also be skipped, as some of them can work directly on the edge response data (see section 8.2).

8.1 Line Fitting Methods

A basic approach to finding lines in binary images with extracted edge pixels is to fit lines to the edge pixels. This section introduces two different methods for solving the line fitting problem. First, a very general approach based on the discrimination of outliers and inliers is introduced. This approach is followed by a voting method that uses a certain parameter space to find related points in the image.

8.1.1 RANSAC

A very general approach is to use the iterative method of Random Sample Consensus (RANSAC) [59]. First, a hypothesis of a model is generated by randomly selecting enough sample points from the given data. Then, the hypothesis is evaluated by testing for support from other samples. If an outlier was chosen for the hypothesis, the support from other samples will be very small. This process can be repeated until

a model with enough supporting samples is found.

This approach can be adapted to find multiple models by sorting out samples of identified models and repeating the search until no more models are found in the remaining data. For our problem, the model is a simple line l that can be defined by two sample points from the edge pixels in the image:

$$l(s) = p_1 + s * (p_2 - p_1) = p_1 + sv \tag{8.1}$$

The support of a line hypothesis is estimated by counting all pixels whose distance to the line is less than a given threshold t:

$$Supp_l = \sum_i^N \begin{cases} 1 \; if \; dist(l, p_i) < t, \\ 0 \qquad\qquad\qquad else \end{cases} \tag{8.2}$$

As input, the algorithm needs a data set D of points given by the edge pixels, the number of maximum iterations i to find a line, the threshold t to estimate the inliers, and the number g to distinguish good model fits from false lines by requiring that $Supp_l > g$. The basic algorithm then works as follows

1. Randomly select two points from D and create a line model
2. Find inliers. If there are enough ($> g$) save the model
3. Repeat from 1. until maximum iteration is reached
4. If no model is found, stop the algorithm, otherwise add the line model with the most supporting points to the line list, remove points from D and continue with step 1.

Many optimizations are possible. Instead of always looping through all iterations, step 4 can be reached as soon as a valid model is found. The criteria for a valid model can be extended, e.g. by adding a density constraint that discards models with enough inliers, but sparsely distributed around the model. A line can be re-fitted by all supporting points. The error of the fit can then be used to select the line with the least error from the list of accepted lines in step 4.
The RANSAC method is easy to implement and can be applied to many model fitting problems. However, additional processing is required to extract the final line segments, and the computational cost increases rapidly with a high rate of outliers. In the case of line detection, the cost explodes for large images. This can be handled by reducing the number of image pixels by combining RANSAC with a probabilistic approach, or by creating subsets, e.g. by applying the Hough transform.

8.1.2 Hough Transform

The concept of the Hough transform is to describe a geometric object by an equation with two or more parameters that form the basis of the parameter or Hough space. A single point in image space (considered in isolation) can lie on an infinite number of these geometric objects. Typically, the Hough space also discretizes the possible number of objects because it is represented by an n-dimensional map (depending on the number of parameters) that has a fixed and discrete range for each dimension. Given this map, a single point votes for a fixed number of possible geometric objects in Hough space. Accumulating all the votes in the Hough space from all the detected

edge points in the image space, gives a distribution of the geometric object parameters that indicates the most likely geometric objects at the local maxima's.

The Hough transform is easily applied to lines. After Paul Hough laid the foundation for the Hough transform with his patent in 1962 [93], Rosenfeld gave the first real definition of the Hough transform applied to lines in his book on computer vision [181]. He defines the transformation algebraically as a point-slope problem:

$$y = y_i x + x_i \tag{8.3}$$

where (x_i, y_i) are the edge points in the image. If a set of points (x_i, y_i), $i = 1 \dots n$ are collinear, the corresponding lines will all intersect at a single point in the Hough space. But if the points (x_i, y_i) are on a line that is nearly parallel to the x-axis, then the corresponding lines in Hough space will all be nearly parallel, resulting in an intersection point that recedes to infinity. Rosenfeld solved this problem by scanning the image twice, swapping x_i and y_i for the second scan.

Motivated by mathematical approaches based on integral geometry, [163] Duda and Hart [47] modified the interpretation of the Hough space for lines given by Rosenfeld. It is based on the statistics of random geometric events used to characterize properties of shapes in images. To randomly throw a line on finite subsets of a plane, a suitable probability space must be found. This is done by introducing an invariance condition for the geometric objects. Duda et al. required that the results of the random line tossing calculation be invariant to translation and rotation of the geometric figures. This resulted in a distance and angle or normal parameterization of a line:

$$y = -\frac{x\cos\theta}{\sin\theta} + \frac{d}{\sin\theta} \tag{8.4}$$

The normal equation of a line also solves the problem of the theoretically unbounded transformation space as proposed by Rosenfeld. The bounds of the parameter space can be defined from the image space $= 0 \dots 2\pi$, $d = 0 \dots d_{max}$, while d_{max} can be determined as the diagonal of the image space:

$$d_{max} = \sqrt{width^2 + height^2}. \tag{8.5}$$

A point (x_i, y_i) in image space can be mapped to the curve in $d - \theta$ parameter space:

$$d = x_i\cos\theta + y_i\sin\theta \tag{8.6}$$

With this transformation, points in a finite image space are mapped to sinusoids in a finite transformation space, and points along a line are mapped to intersecting sinusoids, regardless of line orientation or choice of coordinate axes.

The relationship between image space and parameter space can be summarized as follows:

1. A point in the Hough space corresponds to a line in the image space (first assumption of the Hough transform).
2. A point in image space corresponds to a sinusoid in Hough space.
3. Points on the same line in image space correspond to curves in Hough space that share a common intersection.

4. Points on the same curve in parameter space correspond to lines in image space that have a common point of intersection.

Basic Algorithm
The basic algorithm proposed by Duda et al. can be summarized in four steps:

1. Compute edge points (e.g. Canny)
2. Initialize H[d,θ]=0 (accumulator)
3. for each edge point $I[x_i, y_i]$ in the image for $\theta = 0$ to 180

$$d = x_i cos\theta + y_i sin\theta$$

$$H[d, \theta] += 1$$

4. Find the value(s) of (d,θ) where H[d, θ] is maximum

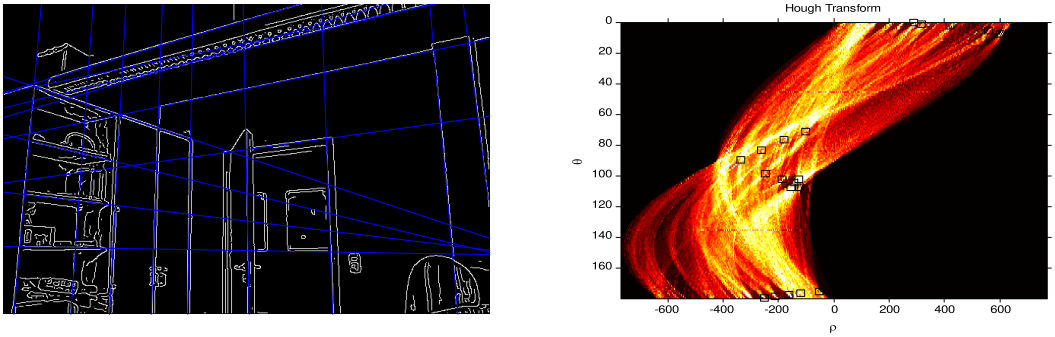


FIGURE 8.1: Example of a Hough transform. The left image shows the edge pixels and the detected lines. The right image shows the Hough transform of the given edge pixels.

Similar to RANSAC, the Hough transform works on unconnected edge pixels and is therefore insensitive to noise, gaps, and occlusion. It can also be applied to different and more complex models. For large images or high precision lines, the computational cost and memory requirements are high. Additional processing is required to extract the final line segments, such as tracing the line and searching the edge map for connected components that support a particular line segment.
Since the Hough transform is a well-known method in computer vision, many extensions and adaptations exist. Some basic extensions are

- Instead of incrementing a single point in the accumulator, an entire patch can be incremented to handle imprecise edge locations.
- The votes in the accumulator can be weighted to favor strong edges.
- The gradient of the image can be used to get an initial estimate of the edge orientation. In this case, the range for θ can be reduced.

Princen et al. [175] proposed a hierarchical approach to the Hough transform. It is combined with a pyramid structure. For each layer in the pyramid, the image is divided into several sub-images. The top level image is the whole image. The algorithm starts at the bottom layer to detect short line segments using the Hough

transform on small sub-images. The results are propagated up the pyramid until the top layer is reached.

Illingworth et al. [98] used an adaptive approach by first using a coarse accumulator to locate areas of interest and then refining these areas until a specified accuracy is reached.

Xu et al. [231] introduced a randomized Hough approach. Similar to RANSAC, two points are randomly selected to form a line. Along the line, nearby points are collected and used to get a final vote for the line. This is repeated to find the global maximum. Finally, the pixels of the selected line are removed and the whole process is repeated.

Kiryati et al. proposed a probabilistic approach. Instead of using all N edge points, only a subset n (n≪N; 15%-25%) is used. Intuitively, this works because a random subset of N will fairly represent all features and noise. A more recent and faster approach was developed by Fernandes et al. [56]. They used a cluster of approximately collinear pixels to cast votes in the accumulator. The votes are based on oriented elliptical Gaussian kernels that model the uncertainty associated with the best-fitting line with respect to the corresponding cluster. This approach produces a cleaner accumulator map and is more robust to false line detection.

Recently, a sophisticated Hough-based method that can extract high quality line segments even in noise is introduced by Guerreiro et al. [77]. However, the quality results in high computational cost, making the approaches not attractive for real-time applications.

This is only a brief overview of existing extensions. Much more literature on this topic can be found on the Internet [97, 15, 9, 121, 197, 76, 25, 217]. A combined approach of RANSAC and Hough transform is introduced in the next section.

8.1.3 Clustered Random Hough

As mentioned above, the computational cost of the RANSAC approach increases rapidly with a high rate of outliers, making it unattractive for finding lines in images. On the other hand, the Hough transform is quite expensive for large accumulator maps, which are needed to accurately determine the lines or to distinguish between multiple nearby lines in an image. A combination of these two methods is straightforward, since a coarse Hough transform can be used to generate reasonable subsets or clusters of edge points at low cost, which can be efficiently processed with RANSAC.

The basic algorithm consists of the following steps:

1. Create a coarse accumulator map for the Hough transform. Each entry of the map is defined as a bin, which can contain an arbitrary number of points. The size of the map is defined by a fixed size ds for the distance and a fixed size os for the orientation:

$$H[d,\ \theta], \quad d \in \left[0, 1, \ldots, \left\lceil \frac{d_{max}}{sa} \right\rceil\right],\ \theta \in [0, 1, \ldots, \left\lceil \frac{\pi}{os} \right\rceil \tag{8.7}$$

2. Precompute sine and cosine for all $\theta_i * \left\lceil \frac{\pi}{os} \right\rceil$

3. Fill the accumulator bins by iterating through the edge pixels: For each pixel, compute the quantized distance and orientation, and add the point to the corresponding bin. To avoid fragmentation, unquantized values near a quantization boundary also vote for the other bin. Store the associated bins for each point in an additional map
4. For a bin with enough associated points, run RANSAC to detect lines.
5. For each line detected, remove support points from the bin and adjacent bins.
6. Continue with the next bin at step 4 until all bins have been processed.

A comparison of this approach with other methods is presented in the last section of this chapter.

8.2 Edge Region Methods

The edge region methods operate directly on the edge responses provided by the methods introduced in chapter 4. They use edge orientation and edge strength to create clusters or support regions for line segments.

8.2.1 The Burns Method

Burns et al.[29] presented an approach in which the extracted gradient pixels are clustered or binned into a fixed set of directions. Then, connected gradient pixels are searched within each bin. The basic algorithm is as follows:

1. Compute the gradient (e.g., Sobel) of an image and the direction for each point above a given threshold of magnitude.
2. Assign each accepted point to one of 8 (or more/less) directional bins.
3. For each direction bin, find connected components of pixels and store them as line support regions
4. Compute attributes of line support regions, such as endpoints, length, straightness, contrast, width, and orientation
5. Filter on attributes

The resulting line regions suffer from two main problems:

1. Over-merging: Visually distinct line regions that are spatially contiguous may be incorrectly merged into a single segment because they have similar orientations and (partially) fall into the same orientation bin. A simple solution to this problem is to use smaller orientation bins (see Figure 8.2).
2. Fragmentation: A line may produce fragmented support regions if the distribution of gradient orientations happens to cross a bin boundary (see Figure 8.3). In this case, the simple solution would be to use fewer bins.

The simple solutions are contradictory, so an advanced solution to one of the two problems that is independent of the other is needed. Burns et al. suggested using bins that overlap by 50%, e.g. bins of 45° starting at 0° and 22.5°. When a line crosses

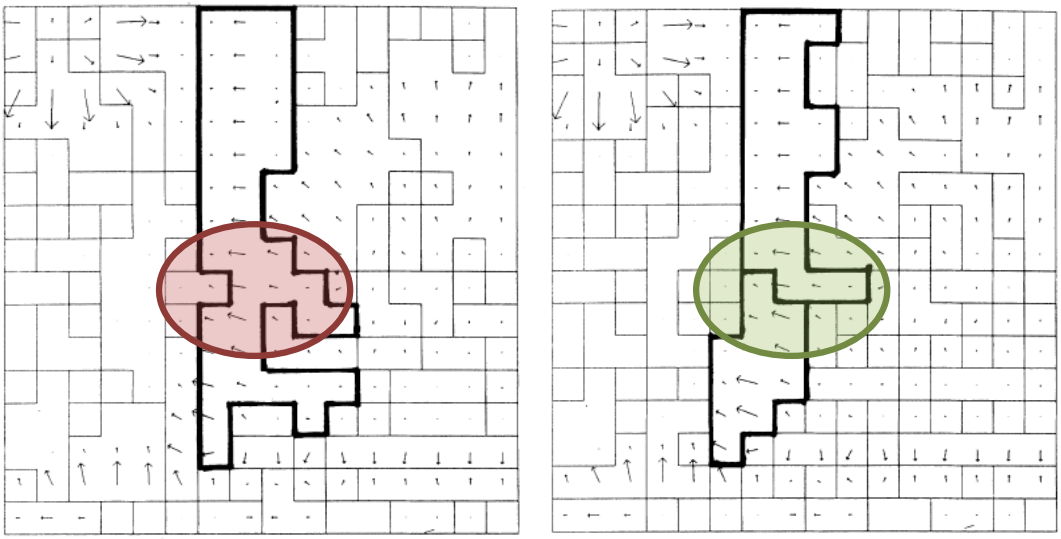


FIGURE 8.2: Example of over-merging. Left - Too large bins, region is merged. Right - Smaller bins, region is not merged.

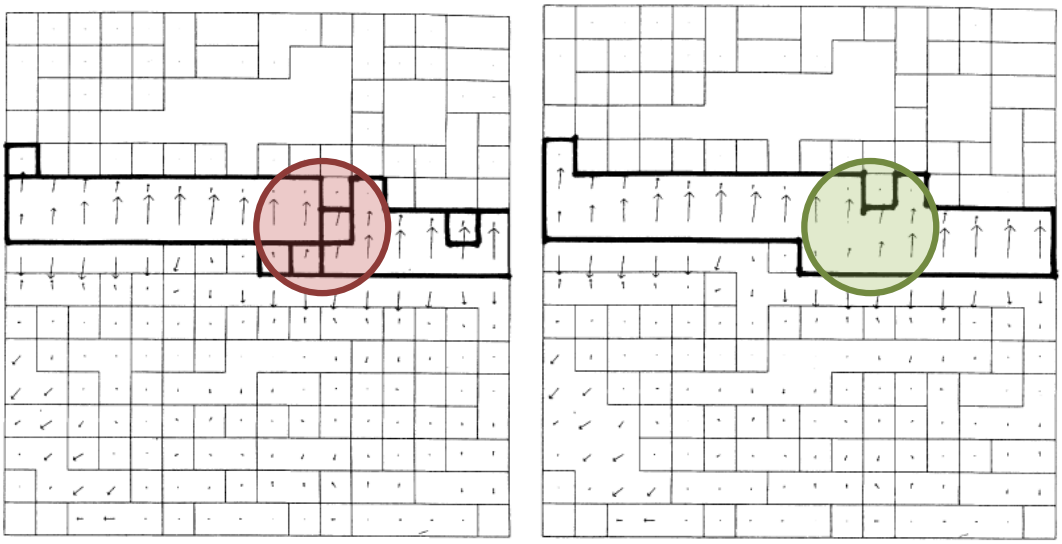


FIGURE 8.3: Example for fragmentation. Left - Gradient orientations lie across a bin boundary, Right - Gradient orientations are within the bin range.

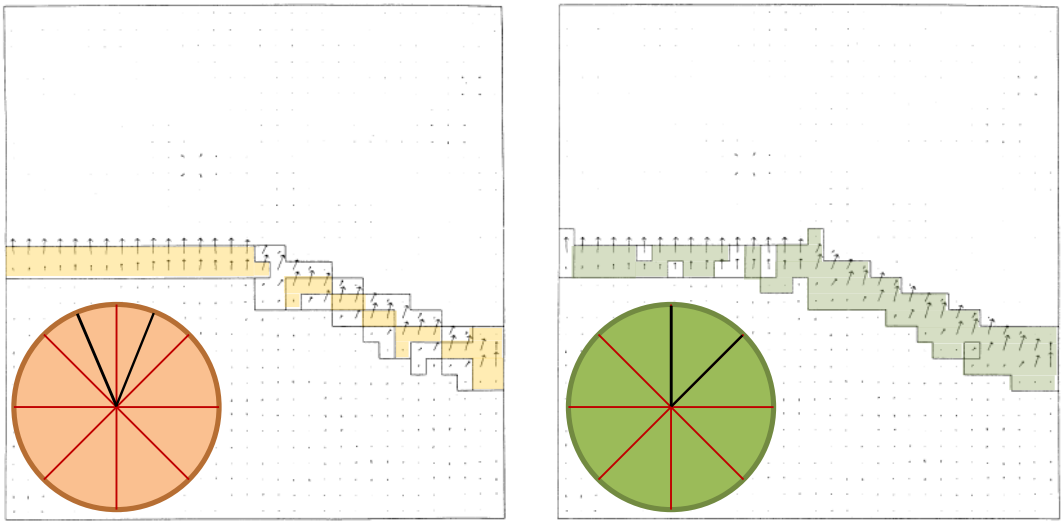


FIGURE 8.4: Example of 8 directional bins with two cases: Left - Example of a bin with 50% overlap with two other bins. The colored area shows the corresponding edge responses that fall within the bin range (all points with an orientation within the bin range are selected), Right - Example of a normal bin with corresponding edge responses.

a bin boundary, the corresponding overlapping bin does not tend to fragment the same line. So each edge point falls into two bins and therefore has two line support regions (see Figure 8.4). Now the problem arises how to assign edge points to the line support region of a bin. The presented solution makes the following assumption: a bin fits best if the length of the line support region in this bin is the longest. The algorithm can be described in four steps:

1. Compute line support region lengths for overlapping bins
2. Each edge point votes for the longer line support region
3. Each line support region gets a vote count
4. If the number of votes for a line support region is less than 50% of the number of line support edge points, the region is discarded.

The final line support regions can be used to estimate the line segments using a line fitting method as discussed in section 8.1.

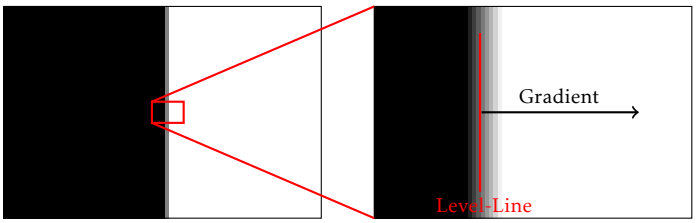
8.2.2 Gioi Line Segment Detector (LSD)

Another more recent method was introduced by Gioi et al. [67]. They used a kind of flood-fill approach based on edge pixel orientation to extract line support areas (see Figure 8.5). The basic algorithm is as follows:

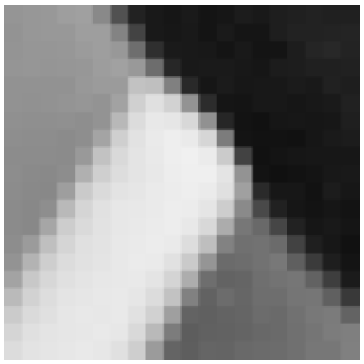
1. Compute the image gradient (they suggested using the Roberts operator, see section 4.1) with perpendicular direction, the level line (see Figure 8.5):

$$\theta = atan2(gx, gy)$$

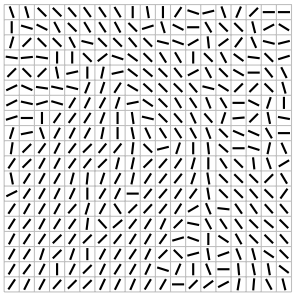
2. Generate seed points by sorting them into bins according to their magnitudes



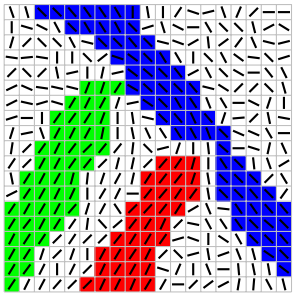
(A) Level-Line Extraction from Gradient



(B) Grayscale Image



(C) Extracted Level-Lines



(D) Line Support Regions

FIGURE 8.5: This figure illustrates how the LSD algorithm works. (a) shows the extraction of level lines from the gradient image. (b) A highly magnified example of a grayscale image. (c) shows the extracted level lines based on the image gradient. (d) shows the line support regions separated by color.

3. Use a threshold ρ to reject pixels with low magnitude. The threshold is derived from the quantization error q and the angular tolerance τ for the flooding process:

$$\rho = \frac{q}{sin\tau}$$

4. Perform region growing starting from an unused seed point with orientation θ_r. Use 8-connected neighborhood to find adjacent pixels with similar orientation ($|\theta_r - \theta_p| < \tau$). After adding a pixel, update the region direction based on all orientations of the region supporting points:

$$\theta_r = atan2\left(\sum_j \sin\left(\theta_j\right), \sum_j \cos\left(\theta_j\right)\right)$$

5. Compute oriented bounded rectangles for regions (see section 8.4 for more details)

6. Validate the support region using NFA. Try to improve the region rectangle if it fails the NFA test before discarding it completely.

7. Detect low density region rectangles and refine region

Steps 1 through 5 are straightforward and have been covered in other sections. However, the last two steps are worth a closer look. Although they could be skipped for performance reasons, they can greatly improve the final result. In section 7.4 the validation process of a line segment is already discussed. Gioi et al. have slightly

adapted the NFA test for their rectangle support region:

$$NFA(r) = (NM)^{\frac{5}{2}} \gamma \bullet \sum_{j=k}^{n} \left(\begin{array}{c} n \\ j \end{array} \right) p^j (1-p)^{n-j} \tag{8.8}$$

The variables N, M represent the rows and columns of the image, while γ is the number of different precision values p potentially tried (initial is $p = \frac{\tau}{\pi}$) in the region improvement process. The variable n defines the total number of points in the region, while k is the number of aligned pixels (up to a tolerance of $p\pi$) to the region orientation computed in step 5. The main difference to the previously discussed approach is the number of potential segments, which is adapted to rectangles and multiple tries with several precision levels. The multiple tries are derived from the improvement process. If the NFA test fails, the first 5 improvements are tried before the region is discarded.

1. Try finer precision: $p/2$, $p/4$, $p/8$, $p/16$, and $p/32$.
2. Try to reduce rectangle up to 5 times with a 0.5 step
3. Try to reduce one side of the rectangle (same as step 2)
4. Try to reduce the other side of the rectangle (same process as step 2)
5. Try again five finer resolutions with even smaller values: $p/64$,...

If a region passes the NFA after a refinement step, the region is accepted and the refinement is aborted. The maximum precision attempts result in a number of precision values of $\gamma = 11$.

Step 7 addresses the problem of over-merging as introduced in the previous section. To detect over-merging regions, a density of aligned points k is calculated based on the rectangle width and height:

$$d = \frac{k}{length(r) \bullet width(r)} \tag{8.9}$$

If the density is less than a given threshold D (the default value is 0.7), two refinement strategies are applied:

1. Reducing angle tolerance τ: a new tolerance τ' is estimated by calculating the standard deviation σ of the orientations near the seed point. Then the new tolerance is set to $\tau' = 2\sigma$, the state of all collected region points within the given region is cleared, and the region growing is repeated.
2. Reducing the region radius: If the first strategy fails, a second method is tried. In this method, the pixels with the greatest distance to the seed point are gradually removed until the criterion is met or the region is too small and rejected. The distance of the farthest point to the seed is estimated and then reduced to 75%. All points not within this radius are removed.

The validation and refinement steps of Gioi et al. can also be applied to the Burns et al. method to improve the final support regions for line segments. A simple approach to reduce run time is to adapt the methods to work with thin boundary regions, such as those provided by NMS (see subsection 6.2.2). Since the thinning process can be

handled much more efficiently than the connected component search, a shorter run time is achieved in most cases. For the Burns et al. method, step one must be replaced by the gradient and NMS computations. The final line segment estimation is also slightly different, as explained in section 8.4. The same adjustments are required for the LSD method. The validation and refinement procedures are still valid for thin regions, but less effective. Problems at junctions can be more severe, since NMS is more sensitive to poorly oriented edge pixels.

More approaches exist, e.g. by using clustering methods to extract line segments [218, 234, 12] or by applying some kind of swarm intelligence to extract line segments [107].

8.3 Linked Edge Methods

Based on the results of the connected edge pixels or edge primitives from chapter 7, several methods for line segment extraction can be used. As input, a connected chain of edge pixels or edge primitives is expected, and as result, the chain is divided into straight line segments depending on some criteria such as minimum length or maximum distance of support pixels to an estimated line.

8.3.1 Least Square Fit and Extend

A method proposed by Akinlar et al. [2] is to walk over the pixel sequence and fit lines to a fixed subset using a least square fit method. After finding a subset with a small error, the set is expanded until there are no points in the sequence with a small distance to the fitted line. Thus, the algorithm consists of two main parts to find a fit. If the sequence is split, the same procedure is called recursively with the remaining list of points:

1. Computes a least-squares fit of a subset of the pixel sequence with a given minimum length l_{min} at position i (initial $i = 0$). If the resulting error e is below a given threshold t, a valid fit L is found, so continue with step 2. As long as $e > t$, shift the start of the subsequence by $i = i + 1$ and repeat the fit until the remaining length of the shifted subsequence is less than the given minimum length. If no fit is found, the sequence is discarded.

2. Start at position i with length $l = l_{min}$ and given fitted line L and repeat until the end of the subsequence is below the end of the full sequence. Calculate the distance d between the pixel and the line. Use the pixel at position $i + l + 1$. If the distance is $d < t$, increment the length by $l = l + 1$, otherwise exit the loop.

3. Recursively start at step 1 with the remaining pixel sequence

Intuitively, the algorithm extends the line segment pixel by pixel to a found line fit until the end of the sequence or a corner is reached. A corner causes a large distance between the line and the supporting pixel, forcing the process to split the sequence.

The authors suggest to use a threshold $t = 1$ and a minimum length based on the probability of a meaningful line depending on the image size. For a line of minimum length, the NFA is valid only if the number of aligned pixels k is equal to the number of total pixels n of the segment:

$$NFA(n) = N^4 * p^n \tag{8.10}$$

Now n can be determined as:

$$n \geq \frac{-4\log(N)}{\log(p)} \tag{8.11}$$

The precision $p = \frac{\tau}{\pi}$ depends on the angle tolerance, which is set by default to $\tau = 22.5°$, so $p = \frac{\pi}{8\pi} = \frac{1}{8}$.

Depending on the tolerance t, segments with high curvature are split into many small line segments or rejected if the minimum length is not reached. Using a higher tolerance allows curved segments to be interpolated with fewer line segments, but may result in small over-interpolation areas. The dependency on a minimum length to perform a line fit can result in missing fine structures in the image. While a line segment may not be meaningful because it is too short, the initial pixel sequence may be very meaningful.

Adaptive Least Square Fit and Extend

While the original Least Square Fit and Extend method only adds points to the initially found line fit until all points are added or the distance tolerance is violated, the adaptive version always updates the line fit and reevaluates the distance to all line support points. In this way, the line fit evolves as new points are added, automatically correcting initial bad fits and giving more tolerance to slightly "curvy" edge segments.

8.3.2 Split and Merge

Another approach with slightly more control over the split locations is based on finding the maximum distance with respect to a baseline. A method that can perform such a task was introduced by Ramer [177] and at about the same time by Douglas and Peucker [45]. In the literature it is often referred to as the Douglas-Peucker algorithm, the iterative end-point fit algorithm, or the split-and-merge algorithm (see Figure 8.6).

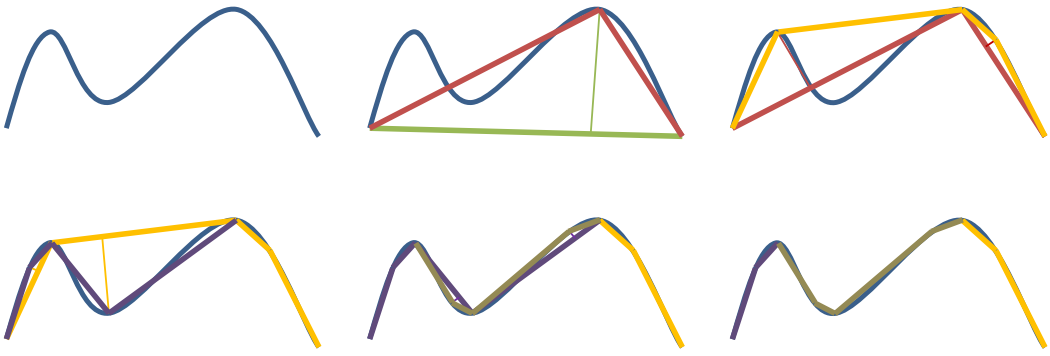


Figure 8.6: This graphic illustrates the steps of the Ramer or Douglas-Peucker algorithm, which simplifies a curve by approximating it with linear segments. The process begins by connecting the first and last points, then recursively identifies the key points that deviate the most from the straight-line approximation. The final result is a reduced set of line segments that closely represent the original curve while minimizing complexity.

Starting from a given curve composed of line segments, the basic concept of this method is to reduce the linear parts so that the original curve is still represented

up to a given distance error. The simplified curve consists of a subset of the points that defined the original curve. To apply this scheme to a sequence of pixels, it is interpreted as a curve with many small linear parts defined by the neighboring pixels. The recursive algorithm works as follows:

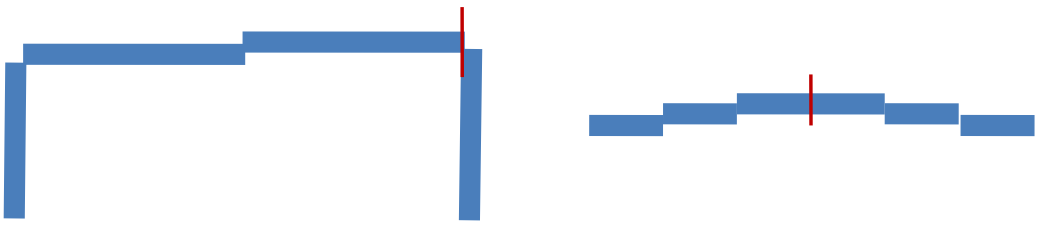
1. Use the endpoints of the given pixel sequence $\boldsymbol{p}_b, \boldsymbol{p}_e \in S$ to calculate the baseline:

$$\boldsymbol{n} = \begin{bmatrix} -\Delta y \\ \Delta x \end{bmatrix} = \begin{bmatrix} \boldsymbol{p}_{by} - \boldsymbol{p}_{ey} \\ \boldsymbol{p}_{ex} - \boldsymbol{p}_{bx} \end{bmatrix}, \quad \boldsymbol{n}_0 = \frac{\boldsymbol{n}}{|\boldsymbol{n}|}, \; d = \boldsymbol{n}_0^t \boldsymbol{p}_b$$

2. Calculate the distance to the baseline for all points in the sequence and store the maximum distance as $d_{max} = \max\left(d_{max}, \left|\boldsymbol{n}_0^t \boldsymbol{p} - d\right|\right) \forall \boldsymbol{p} \in S$ and index i of that maximum in the sequence.

3. If the maximum is above a given threshold $d_{max} > t$, split the segment at index i and start both sequences over at step 1. Otherwise, mark the subsequence as a separate sequence.

The basic algorithm does not take into account that a small adjacent subsequence of pixels can generate multiple identical maxima. In this case, an additional rule is needed to decide where to split the sequence if $d_{max} > t$ (see Figure 8.7). Two cases can be considered:



(A) (B)

FIGURE 8.7: (a) Split rule for hard corners, (b) Split rule for curved contours.

1. A sequence of line segments with hard corners containing a sequence of maximum pixels: in this case, the "stronger" corner adjacent to the sequence of maximum pixels must be detected and selected for splitting.

2. A sequence of line segments approximating a curved contour: In this case, there is no preferred "stronger" corner, so the center of the maximum sequence must be used for splitting.

Compared to the previous approach, no minimum length is required to find a first estimate of the line to be processed. Depending on the maximum distance (in pixels), the granularity of the approximated line segments can be controlled. From experimental results, a distance threshold $t = 2$ is proposed.

One problem with this method is ill-formed segments. This can lead to unfortunate splits on straight parts of edge segments. While this can be fixed by applying an additional merge rule after the splitting, the extra overhead can be saved if ill-formed segments were already prevented before the splitting. A simple method to reduce

ill-formed segments is to enable the corner rule within the edge linking process as discussed in section 7.2.

Relaxed Error Distance

Another problem comes from long straight edge segments, especially along weak edges. They tend to be poorly localized, resulting in gaps and unwanted short directional anomalies. The gaps are handled in the segmentation process, but the directional anomalies can lead to unwanted splits in the split and merge process. To reduce the probability of such splits, the error distance parameter can be adjusted locally. If a found maximum distance is greater than the given threshold, the local gradient is analyzed and used in combination with the length of the two resulting line segments to relax the threshold. If the distance is still larger, the segment is split.

The two thresholds $t_1,\ t_2$ indicate the normal error distance (t_1) and the maximum relaxed error distance (t_2). If the computed maximum distance $d_{max} > t_1$, the relaxed threshold is determined locally as:

$$t_r = t_1 + (t_2 - t_1)\sqrt[3]{w_1 w_2 w_3};\ \ t_r \leq t_2 \tag{8.12}$$

The variables w_1, w_2 and w_3 are weighting factors defined as follows:

$$w_1 = 1 - \frac{m_{mean}}{m_{max}} \tag{8.13}$$

Where m_{mean} is the local mean size at the location of the maximum distance (3x3 region) and m_{max} is the maximum size value within the image.

$$w_{2/3} = \frac{length(l_{1/2})}{l_{max}} \tag{8.14}$$

Here $l_{1/2}$ are the two lines defined by the difference of the segment endpoints and the point of the maximum distance found. The reference length l_{max} is determined by the image size:

$$l_{max} = \frac{diag(img)}{3} \tag{8.15}$$

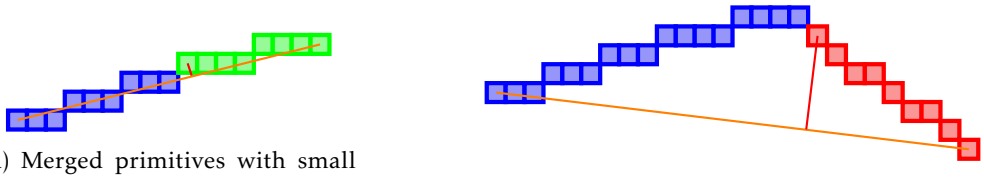
All values above 1 are truncated to 1, and the values below 0 are set to 0. Thus, the relaxation is maximal for split positions with small size and two long resulting line segments.

8.3.3 Primitive Patterns

For the primitive pattern sequence, the first approach can be simplified because a direction is already provided by the pattern. So only the merge step is applied:

1. Use the first pattern in a sequence to estimate a line for the sequence
2. Compute the distance between the adjacent pattern and the estimated line. If the distance is below a given threshold, continue with the next pattern in the sequence, otherwise split the pattern sequence and continue with adjacent patterns

If the initial pattern is a poor line estimate, splitting may occur in an awkward range, resulting in non-optimal line segments. A more robust approach is to update the line



(A) Merged primitives with small distance.

(B) Merged patterns with large distance.

FIGURE 8.8: This figure shows two merge examples of edge pattern with small and large error. Depending on a given threshold, the patterns are merged into a single pattern or split again and stored as two patterns.

estimate after each new pattern added to the sequence, e.g. by using the endpoints (see Figure 8.8). For short patterns, this can also lead to poor line estimation, resulting in over-merged segments. In this case, the distance to the new line estimate must be checked for all pattern transitions to ensure that no distance grows larger than a given threshold and that the new estimate still fits all included patterns. An alternative would be to use the refinement method proposed by Gioi et al.

An adaptation of the split-and-merge method can also be used for the pattern sequences. Instead of calculating the distances for all pixels within a given sequence, only the pixels at the pattern boundaries are used, since the maximum distance for a direction change can only be found at these points. It is recommended to use this approach, as it again gives better control over the splitting process without any real difference in performance.

To fit a line segment through a given support region, different approaches can be used, depending on the required accuracy and run time. For RANSAC or Hough-based approaches, the support regions must be extracted before the line segments can be extracted. A basic algorithm can be implemented as follows:

1. Sort the support points of a line by their distance from the line origin parallel to the line direction. If the support points are not available, collect them by tracing the line.
2. Calculate the maximum length of the line segment
3. Split the line segment for large gaps between two directly adjacent points
4. Discard short or meaningless line segments that result.

Given a line support region as a sequence of pixels, as provided by edge linking methods, the simplest approach to estimating the line segment is to use the endpoints of the sequence. In most cases, this will lead to a poor estimation, resulting in poorly fitting line segments, or worse if an outlier has been selected. To be more robust to outliers, the centroids of each half of the sequence can be used to estimate the line direction:

$$c_1 = \left(\frac{2}{n} \sum_{i=1}^{n/2} x_i, \frac{2}{n} \sum_{i=1}^{n/2} y_i \right), \quad c_2 = \left(\frac{2}{n} \sum_{i=n/2}^{n} x_i, \frac{2}{n} \sum_{i=n/2}^{n} y_i \right) \tag{8.16}$$

Next, the distances to the line origin of the end points of the support sequence, parallel to the line direction, must be calculated to complete the line segment:

$$d = n_{0y} p_x - \quad n_{0x} p_y \tag{8.17}$$

8.4 Line Segment Fitting Methods

Several methods can be used to fit a line segment to a given edge segment. The most common methods are Linear Regression and Principal Component Analysis (PCA). Both methods use the least squares method to minimize the distance between the line and the support points. The main difference is that linear regression only minimizes the distance in one direction, while PCA minimizes the distance in both directions. The PCA method is more robust to outliers, but also more expensive. This section briefly introduces both methods.

8.4.1 Linear Regression

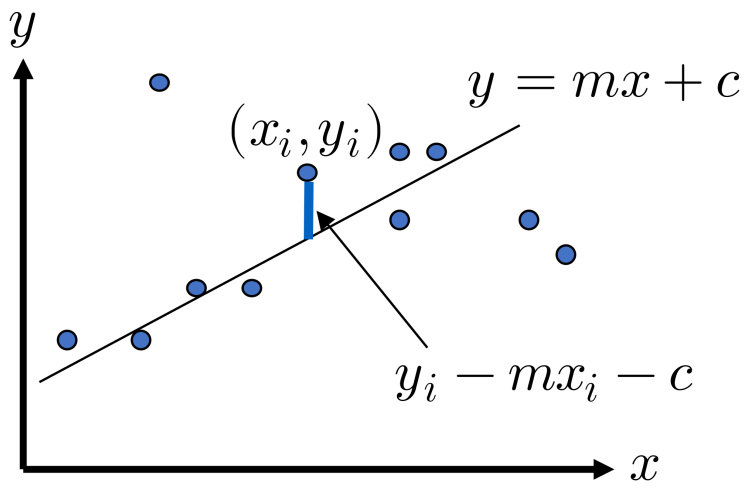


Figure 8.9: This figure illustrates a linear regression model where a best-fit line is drawn through a set of data points. The line represents the relationship between the independent and dependent variables, minimizing the total error (residuals) between the actual points and the predicted values. The goal of linear regression is to find the optimal slope and intercept that best explain the trend in the data.

The line segment estimation methods introduced so far do not produce an optimal fit to the line support regions by minimizing the mean error. To fit a line segment by a minimized mean squared error, linear regression [46] can be applied. For a line given by m and c, the following squared error function can be defined:

$$E = \frac{1}{n}\sum_{i=1}^{n}(y_i - mx_i - c)^2 \tag{8.18}$$

The error function is a 2D paraboloid. Thus, the minimum error distance for a line within a given point list can be found by estimating the line parameters of its derivatives:

$$\frac{\partial E}{\partial m} = 0 \; and \; \frac{\partial E}{\partial c} = 0 \tag{8.19}$$

To find a point within the point set with minimal error, the centroid can be determined:

$$c_m = (\overline{x}, \overline{y}); \;\; \overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i, \; \overline{y} = \frac{1}{n}\sum_{i=1}^{n} y_i \tag{8.20}$$

The centroid is a point that lies on the line with the least error. It can be used as a reference point for the line:

$$c = \overline{y} - m\overline{x} \tag{8.21}$$

The slope can be estimated from the correlation of the variances:

$$m = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sum_{i=1}^{n}(x_i - \overline{x})^2} \tag{8.22}$$

Then, the normal and distance of a line can be calculated as:

$$\begin{gathered} \boldsymbol{n} = \begin{bmatrix} -\sum_{i=1}^{N}(x_i - \overline{x})(y_i - \overline{y}) \\ \sum_{i=1}^{N}(x_i - \overline{x})^2 \end{bmatrix}, \ \boldsymbol{n}_0 = \frac{\boldsymbol{n}}{|\boldsymbol{n}|} \\ d = c \bullet \boldsymbol{n}_0 \end{gathered} \tag{8.23}$$

To avoid the singularity for vertical lines, the line estimation is switched as soon as $(y_i - \overline{y}) > (x_i - \overline{x})$:

$$\begin{gathered} c = \overline{x} - m\overline{y}, \\ m = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sum_{i=1}^{n}(y_i - \overline{y})^2} \end{gathered} \tag{8.24}$$

This approach minimizes the root mean square error using only one axis. If the line is strongly vertical or horizontal, the estimation from the line is quite good, but for diagonal lines it is not optimal because both components (x and y) are strongly represented.

8.4.2 Principal Component Analysis

A more accurate solution is to apply the eigenvalue decomposition or principal component analysis (PCA) [100] to extract the principal component axis. This is done by determining the covariance matrix and computing the eigenvalues and eigenvectors from this matrix, e.g. by applying the singular value decomposition (SVD) [223]:

$$cov(\boldsymbol{X}, \boldsymbol{Y}) = \begin{bmatrix} \sum_{i=1}^{n}(x_i - \overline{x})^2 & \sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y}) \\ \sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y}) & \sum_{i=1}^{n}(y_i - \overline{y})^2 \end{bmatrix} \tag{8.25}$$

The covariance matrix contains information about the dispersion of the data (top left and bottom right in the matrix) and the correlation (covariance) between the components (top right and bottom left in the matrix). The resulting eigenvalues represent the mean square deviation for the two components, depending on the corresponding eigenvectors, which represent the orientation of the deviation. For a line, the eigenvector of the stronger eigenvalue represents the line direction. The other eigenvalue should be much smaller than the first, since the second component should be very weak for an expressive line.

Since this matrix is of symmetric nature, the eigenvalues and eigenvectors can be calculated directly [73, chapter 5]:

$$
\begin{gathered}
A = \begin{bmatrix} a & b \\ b & c \end{bmatrix} \\
a = \sum_{i=1}^{n} (x_i - \overline{x})^2, \;\; b = \sum_{i=1}^{n} (x_i - \overline{x})(y_i - \overline{y}), \;\; c = \sum_{i=1}^{n} (y_i - \overline{y})^2 \\
\lambda_1, \lambda_2 = \frac{1}{2}(a+c) \pm \frac{1}{2}\sqrt{4b^2 + (a-c)^2} \, If: \\
\lambda = \frac{1}{2}(a+c) - \frac{1}{2}\sqrt{4b^2 + (a-c)^2}
\end{gathered}
\tag{8.26}
$$

The normal for the corresponding line can be determined as:

$$
\boldsymbol{n}_0 = \begin{bmatrix} \frac{b}{\sqrt{b^2+(a-\lambda)^2}} \\ \frac{a-\lambda}{\sqrt{b^2+(a-\lambda)^2}} \end{bmatrix} = \begin{bmatrix} \frac{c-\lambda}{\sqrt{b^2+(c-\lambda)^2}} \\ \frac{b}{\sqrt{b^2+(c-\lambda)^2}} \end{bmatrix}
\tag{8.27}
$$

In the case of line support areas based on rectangular regions (e.g. the LSD method), a weighting can be used to more precisely locate the line within the rectangle, based on a given prior (e.g. the magnitude):

$$
\begin{gathered}
W = \sum_{i=1}^{n} w_i \\
c_m = (\overline{x}, \overline{y}): \;\; \overline{x} = \frac{1}{W}\sum_{i=1}^{n} x_i w_i, \;\; \overline{y} = \frac{1}{W}\sum_{i=1}^{n} y_i w_i \\
a = \frac{1}{W}\sum_{i=1}^{n} w_i (x_i - \overline{x})^2, \\
b = \frac{1}{W}\sum_{i=1}^{n} w_i (x_i - \overline{x})(y_i - \overline{y}), \;\; c = \frac{1}{W}\sum_{i=1}^{n} w_i (y_i - \overline{y})^2
\end{gathered}
\tag{8.28}
$$

where w_i is the weight value at the given location i and W is the sum of the weights used. In this case, the length for the final line segments must be determined by computing the distance from the line origin to all points in the support region parallel to the line direction to find the start and end points in line coordinates.

8.4.3 M-Estimators

A more general approach to estimate the optimal line fit is to use M-estimators [155]. All estimators obtained as the minima of sums of data functions belong to this category, as does the least mean square approach discussed earlier. However, other norms can also be used. An M-estimator is defined as zero by an estimation function. In most cases, the estimator is defined as the derivative of another function. This general definition allows this optimization approach to be applied to diverse and

high-dimensional problems. While a closed-form solution may be available (as in the least squares variant introduced earlier), in many cases an iterative approach is used.

8.5 False Line Segment Detection

For false line segment detection, the same approaches as in section 7.4 can be applied. In general, this step is performed directly on the edge segments before the final line segments are extracted. However, for some methods that provide line segments directly, it may be useful to apply the check afterwards, or in general to find the most meaningful line segments. In this case, the support points for the line segments must be extracted first (if not already present). And for the point locations, the gradient direction must be calculated (if not already available). Then simply apply the NFA test as introduced in section 7.4.

8.6 Line Segment Optimization

Finding line segments with good localization is often of great benefit, as it can help to achieve better final results or improve efficiency. While the line segment estimation discussed in the previous section is an exact fit to the given data, the data itself may not be ideal. It is either a one-pixel-thick point sequence obtained from NMS, ZC and/or linked edges, or it is an oriented support region with width and height obtained from a linked component or flooding process.

For thin pixel sequences, the resulting line segments can be assumed to have a precision of $\pm\frac{1}{2}$ pixels, since the pixels are generally extracted by a local maxima or zero crossing method, which provides the same precision. The precision of the resulting orientation also depends on the length of the sequence, since small sequences based on pixel coordinates have much larger orientation variations. Based on this data, a simple solution to optimize position and orientation is to use interpolated pixels with higher precision, as discussed in section 6.3. However, this information won't be available for thicker line support regions.

The thicker line support regions can provide better data for estimating a line segment because more data is available for regression. However, they take more time to process, and even with weighting, the resulting estimate can be much worse than 1 pixel. The location of the edge within the boundary region is preferably estimated at the strongest slope of the intensity change. Fitting the line without weighting will locate the line at the center of the region, which is not necessarily a good estimate of the strongest slope. The magnitude weighting can help to optimize the location, but if it is poorly distributed, it will result in a poor estimate. Therefore, a true local maxima optimization within this region is preferable.

A third approach to optimizing line segment accuracy is to define a magnitude-based error function that can be used with an optimization method such as the Levenberg-Marquardt algorithm [123, 146]. It requires only an initial estimate of a line segment and iteratively optimizes the line segment by minimizing the error function. There are different approaches using a first-order (e.g. gradient descent [24]) or second-order (or higher) derivative (e.g. Broyden-Fletcher-Goldfarb-Shanno (BFGS) [161]) error function. While the latter methods provide much faster convergence, the runtime cost is generally higher. Therefore, for high-dimensional or large-scale problems, the

first-order derivative approach is often more efficient.

Instead of minimizing an error function, the mean magnitude should be maximized as a function of distance d and orientation θ for a given line segment $l(\theta, d)$:

$$\overline{m}_l(\theta, d) = \frac{\sum_{i=1}^{n} mag(\boldsymbol{x_i})}{n} \tag{8.29}$$

With $n = \lfloor length(l(\theta, d)) \rfloor$ and $\boldsymbol{x}_i \in l(\theta, d)$. Since mag can only return a value for $\boldsymbol{x}_i \in \mathbb{N}^2$, the values for $\boldsymbol{x}_i \in \mathbb{R}^2$ are interpolated using bilinear or bicubic interpolation. To find the values of θ, d where the mean is maximal, the derivatives must be zero

$$\frac{\partial \overline{m}_l}{\partial d} = 0 \; and \; \frac{\partial \overline{m}_l}{\partial \theta} = 0 \tag{8.30}$$

To formulate a minimization problem, which is required for most optimization methods, simply use the inverse $\overline{m}_l(\theta, d)^{-1}$. The derivatives are approximated by finite differences. The search range can be derived from the support region. For an NMS or ZC pixel sequence, a conservative distance range is two pixels $[d-1, d+1]$. The orientation range depends on the segment length:

$$ll = length(l(\theta, d)) \tag{8.31}$$

And the width of the line segment profile on:

$$lw = width(l(\theta, d)) \tag{8.32}$$

A conservative range is specified as:

$$[\theta - \text{atan}\left(\frac{lw}{ll}\right), \theta + \text{atan}\left(\frac{lw}{ll}\right)] \tag{8.33}$$

A quick and easy estimate of the profile width can be derived from the size of the convolution kernel k used to compute the gradient:

$$lw = width(k) \tag{8.34}$$

For thick line regions, the distance range must be adjusted to the range $\left[d - \frac{lw}{2}, d + \frac{lw}{2}\right]$, while the orientation range can remain the same.

Using this approach, a line segment can be optimized for position and orientation by finding the local maximum of the magnitude response averaged over the line segment's support area. The closer the initial line segment is placed to these local maximum, the faster the maximum is found and the lower the probability of converging to a false local maximum.

8.7 Results and Discussion

This thesis implements all the methods described here and makes them freely available with the Line Extraction Framework (see chapter 10). To obtain the final line segments, many steps are necessary, as described in the previous chapters and in this section. They have been designed to be as modular as possible, so that they can be combined with each other, making it easy to implement new or modified methods. In general, there are some approaches that are implemented relatively independently (such as Burns, Hough, or Gioi LSD) and therefore can only be configured to a limited

extent, but there is also the main pipeline of line segment detector processing, in which almost every step can be replaced by other methods. More details can be found in chapter 10.

Before examining the results of the LSD methods, the methods presented in this chapter for decomposing edge segments into straight edge units and for assigning lines to the straight edge units are discussed.

8.7.1 Edge Segment Line Fitting Methods

In contrast to the main section, the line fitting topic is given priority in the discussion. The reason for this is that some of the line splitting approaches have a direct dependency on the line fitting methods. The line fitting methods are used to estimate the line direction and position for some of the split methods. Therefore, it is important to evaluate the line fitting methods first before discussing the edge segment split process.

All variants were tested on the same sets of images, which were selected to cover a wide range of edge types. The images were preprocessed by a Sobel derivative filter (3x3 kernel, see section 4.1) to get the edge responses and the edge points were extracted by a fast nms method using a 8 zone edge direction map (see subsection 6.2.2). Low values for the nms thresholds were selected (lower threshold = 0.004, upper threshold = 0.012), to get a realistic performance (many edge points are present including noise and low contrast edges). Edge segments were extracted using the EdgeLinking method (see section 7.2).

The tests were performed on a high performance HP ZBook 15 Fury G8 notebook with 32GB RAM, Intel Core I7 CPU (11th Gen i7-11850H @ 2.50 GHz, 8 Cores) and NVIDIA RTX A3000 Laptop GPU, running ubuntu 22.04 (jammy). As in previous experiments (see section 5.1), the algorithms were executed based on the datasets BSDS500 and MDB-X.

For this thesis four variants of the line fitting methods were realized and tested:

- **RegressionFit**: Linear regression implementation (see subsection 8.4.1). This method is used to fit a line segment to a given edge segment by minimizing the mean squared error between the line and the support points.
- **EigenFit**: Principal Component Analysis based implementation (see subsection 8.4.2)). This method uses the eigenvalue decomposition of the covariance matrix to find the principal component axis, which represents the line direction with minimal error.
- **EigenCVFit**: Same as EigenFit, but uses an OpenCV based eigenvalue decomposition.
- **MEstimatorFit**: M-Estimator based implementation (see subsection 8.4.3). This method uses an iterative approach to minimize the error function and find the optimal line fit for a given edge segment. It is based on an M-Estimator implementation of OpenCV.

Method	BSDS500	MDB-Q	MDB-H	MDB-F
RegressionFit	0.052	0.103	0.406	1.378
EigenFit	0.055	0.107	0.419	1.438
EigenCVFit	0.969	1.638	6.021	19.751
MEstimatorFit	0.472	0.817	3.037	10.082

Table 8.1: Runtime for line fit methods in milliseconds.

The results of the line fitting methods are shown in Table 8.1. The table shows the runtime in milliseconds for each method. The results show that the RegressionFit method is the fastest, closely followed by the EigenFit method. The EigenCVFit variant is the slowest in this case. This is probably because it is not optimized for so few dimensions and therefore the simply implemented EigenFit for 2D is much faster. The same may be true for the MEstimator variant, which is also much slower than the EigenFit and RegressionFit methods.

Since the differences between RegressionFit and EigenFit in terms of runtime are marginal, it is recommended to use the EigenFit method as it should produce slightly better results. However, this thesis does not provide a more detailed analysis to prove this claim.

8.7.2 Edge Segment Split Methods

This thesis implements all the edge segment split methods described in subsection 8.3.2 and evaluates their performance on the same datasets as the line fitting methods. However, instead of the EdgeLinking method, the EdgePatternLinking method was used to extract edge segments. The results of the segment split methods are shown in Table 8.2. The table shows the runtime in milliseconds for each available method.

The **Ramer** and **ExtRamer** are two different implementations of the same method. While the Ramer implementation is kept as simple as possible, the ExtRamer provides options to change some behavior by swapping template modules at compile time. This is also the reason why there are so many variants of this implementation. Both use the same approach as described in subsection 8.3.2.

LSqr represents the least squares fit and extend methods described in subsection 8.3.1 to fit a line directly to the edge segment and thus split it. It is by far the fastest method (about 6x faster than Ramer), but it is also the least accurate, since it doesn't take into account the shape of the edge segment and may pick non-optimal splitting points, resulting in a bad division of the edge segment into straight sub-segments.

The **ALSqr** variant is a slightly advanced version of LSqr by evolving the line fit even after an initial fit is found (see section 8.3.1 for more details). It is slower than LSqr, but it can avoid bad fits and thus bad splits, especially for "curvy" edge segments.

The **PXRamer** and **PXLSqr** variants are the pattern-based versions of the methods above. The differences are described in subsection 8.3.3. They are generally faster than the simple edge segment variants because they use larger primitives instead of

Method	BSDS500	MDB-Q	MDB-H	MDB-F
Ramer	0.371	0.629	2.362	8.981
Ramer +m	0.394	0.658	2.521	9.602
ExtRamer	0.386	0.650	2.510	9.590
ExtRamer +m	0.401	0.655	2.562	9.919
ExtRamer +es	0.445	0.755	3.023	12.742
ExtRamer +m+es	0.520	0.859	3.458	14.289
LSqr	0.064	0.148	0.622	2.146
ALSqr	0.188	0.812	3.223	10.786
PRamer	0.157	0.250	1.070	4.154
PRamer +m	0.148	0.242	1.056	3.998
PExtRamer	0.156	0.244	1.092	4.179
PExtRamer +m	0.155	0.247	1.030	4.175
PExtRamer +es	0.244	0.383	1.601	7.742
PExtRamer +m+es	0.243	0.369	1.579	7.418
PLSqr	0.123	0.236	0.929	3.412
PALSqr	0.179	0.346	1.368	5.208

Table 8.2: Runtime for segment split methods in milliseconds.

single points to iterate through the edge segments to find critical points to split.

For the **Ramer** methods, the speedup for the pattern variant is quite significant (up to 2x faster), while for the ALSqr variants the speedup is less pronounced, at least for smaller images. For larger images (including more "large" primitives), the speedup factor is up to 2 and may increase for even larger images.

For the **LSqr** variants, I would have expected the same behavior regarding speedup of pattern variant vs. non-pattern variant as for the other methods. But in this case it seems that the compiler is able to highly optimize the processing due to the very compact and direct memory addressing, i.e. the prediction and cache preloading works very efficiently in this case. I leave a more detailed analysis to the enthusiasts who want to understand the behavior down to the last bit.

As mentioned before, there are some processing options, in particular for the ExtRamer implementation:

- **+m**: Stands for the enabled merge option. This option allows to apply a post-processing merge step to merge ill-formed splits, e.g. splits wrongly created by a poorly selected base line.
- **+es**: Stands for the enabled extended split option. In this case, the splitting method uses a relaxed threshold to avoid splits within badly localized areas (see section 8.3.2 for more details).

8.7.3 Line Segment Detectors

With all the methods and approaches introduced and discussed so far, many line segment detectors can be realized. This work provides more than 10 different out-of-the-box line segment detectors that can be used to extract line segments from images. In most cases they offer different options or setups to optimize the approach for a

specific task. In particular, the LSD EL (Edge Linking) method (see section 7.2) has been implemented as a highly modular and configurable approach. Each processing step can be swapped to easily combine a custom detector choosing from almost any of the methods presented in this thesis. See chapter 10 for more details on the architecture.

For the discussion, a set of detectors was selected for a runtime comparison. The settings are the same as in the previous sections.

Method	BSDS500	MDB-Q	MDB-H	MDB-F
LSD EL	3.744	6.744	25.030	94.801
LSD EL Corner	4.090	7.140	26.773	98.786
LSD EL LSSplit	3.648	6.530	24.621	91.644
LSD EP	4.239	7.386	27.523	103.483
LSD EP Corner	4.269	7.526	27.792	104.769
LSD EP LSSplit	4.163	7.372	27.061	101.186
LSD CC	4.408	7.681	29.461	102.463
LSD CC Corner	5.359	9.206	34.338	117.886
LSD CP	4.542	7.950	30.280	107.670
LSD CP Corner	4.745	8.316	31.184	108.777
LSD EDLZ	7.906	14.060	49.167	193.721
LSD FGioi	25.328	50.971	193.583	667.102
LSD Burns	7.078	15.575	58.510	208.862
LSD FBW	12.762	25.336	93.796	301.615
LSD HoughP	25.308	46.590	160.392	515.052
LSD Hough	51.985	106.009	380.554	1,359.215
LSD FBW NMS	5.762	10.514	40.673	144.508
LSD Burns NMS	3.660	7.539	30.693	107.283
LSD EL NFA	4.364	7.295	26.496	105.516
LSD EDLZ NFA	8.567	15.762	55.217	212.864

Table 8.3: Runtime for line segment detector methods in milliseconds.

Table 8.3 shows, that the LSD EL is not only the most modular detector as mentioned before, but also one of the most performant. The LSD EL is about 2x faster than the Burns LSD (see subsection 8.2.1) and EDLZ LSD (original Edge Drawing method, see section 7.1). The Hough method (see subsection 8.1.2) is by far the slowest and produces poor results. In between are the probabilistic Hough (HoughP), the FGioi LSD (see subsection 8.2.2), and the FBW LSD.

The LSD FBW is a fast region growing based approach based on gradient directions. It is similar to the approach of Gioi et al. (see subsection 8.2.2), but simplifies some steps and uses the line fitting methods described in section 8.1 to extract the final line segment of an extracted edge region.

The LSD EP is the pattern-based variant of the LSD EL. LSD CC (connected components) and LSD CP (connected patterns) are simplified, high performance and less modular edge linking implementations. As shown in the table, the LSD EL outperforms the CC variant when the right modules are combined.

The LSD FGioi method always uses an activated orientation based NFA. For the sake of comparability, a variant with active NFA is also listed in the table for LSD EL and LSD EDLZ.

The additional variant names which are also listed in the table have the following meaning:

- **Corner**: Stands for activated corner rule in the edge linking process.
- **LSSplit**: Stands for the Least Square split method instead of the Ramer method.
- **NMS**: Stands for activated NMS for a flooding based approaches. This can significantly improve the runtime, but worsen the quality of the results.

Depending on the problem, the user is free to choose which method is best suited, or even assemble a custom line segment detector. If performance is important, slightly optimized modules that rely almost exclusively on integer operations can be combined to achieve even better performance. If accuracy is important, modules with floating-point support can be used to further increase accuracy at the expense of computation time. Otherwise, specific optimization methods can be used after extracting the line segments. If necessary, only on a specific selection, e.g. by extracting the 20 most meaningful segments in order to save runtime.

Chapter 9

Line Features

This chapter provides a brief introduction to the topic of line features, focusing on descriptors and feature matching. While it does not include an in-depth analysis, it presents key concepts along with some original ideas and perspectives on the subject. The primary analytical work on this topic has been conducted by a colleague, whose research explores the details and evaluation of different approaches. For a more comprehensive examination, including performance analysis and comparative results, please refer to his work [118].

9.1 Point vs. Line Features

Feature extraction and matching is a fundamental task in computer vision problems and has been used in many applications such as object recognition, panorama stitching, object tracking or 3D reconstruction. The basic concept of this approach is to find corresponding features within multiple images of the same scene. This is a complex task because additional changes in image conditions, such as changes in illumination or strong changes in viewpoint, have to be taken into account for feature matching.

Lowe, a pioneer in point-based matching, presented his final Scale Invariant Feature Transform method [135] in 2004 as an alternative to the existing local matching methods based on cross-correlation or sum of squared differences. His approach used efficient rotation invariant point descriptors and could find stable correspondences over a wide scale of image changes, including rotation, scale and illumination changes (See Figure 9.1 for an example). Since then, matching methods based on point features have made great progress and many other efficient approaches have been realized, such as the SURF [13], DAISY [208], BRIEF [30], ORB [183] or AKAZE [5] point feature descriptors.

The main advantage of point features is their geometric simplicity. They can be handled efficiently and there are global constraints, such as the epipolar geometry [83], which can significantly reduce the search area for correspondences. Since they don't have a size, they are also not vulnerable to partial occlusion or fragmentation and have a distinctive appearance for highly textured areas. On the other hand, they also have some significant drawbacks. In most cases, point features along edges are discarded due to their low distinctiveness. While clusters of coplanar point features can be used to uniquely describe an object, they usually do not sufficiently capture the shape or other geometric and topological information. Thus, they fail to detect objects without much textural information, even if they have a significant shape. In

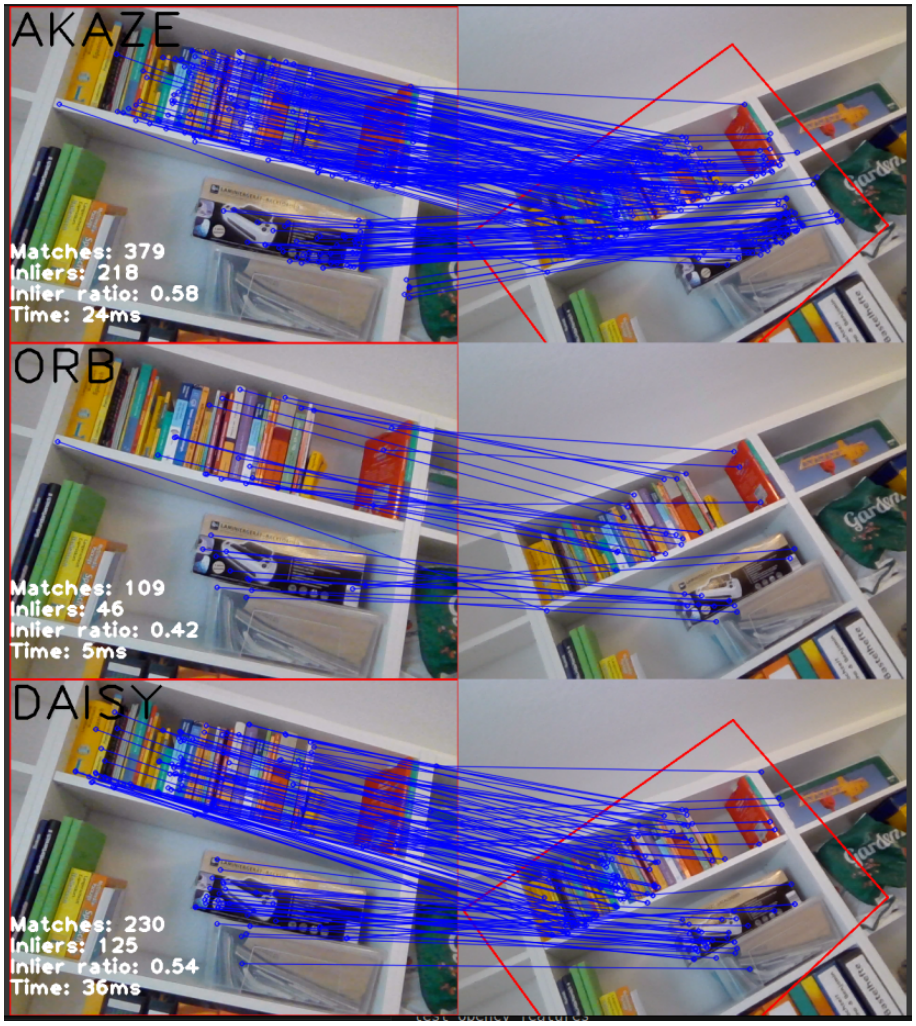


Figure 9.1: Point feature demo including three different point feature descriptors: AKAZE, ORB and DAISY. The image shows the detected features (left) and the matching features (right) between two images of the same scene.

addition, point features are more prone to noise.

Especially in man-made environments, this can lead to problems in applications such as tracking or 3D reconstruction, where objects often lack texture. In this case, line segments are abundant and carry an important amount of geometric and topological information. In addition, they can be used to approximate scene boundaries by representing objects within the scene. Thus, line features can compensate for low texture areas and geometric information, but are more complex in that they have to deal with imprecise endpoints, fragmentation, and less distinctive appearance, while global geometric constraints such as epipolar geometry are not directly available. Nevertheless, line matching is desirable and in some cases essential for many applications.

While point feature matches are usually determined by local similarity alone, line feature matches in most cases also take into account geometric and topological constraints, making them more robust for low texture areas. Line feature matching approaches can be divided into three categories:

1. Local appearance: Similar methods like SIFT are used to determine a local descriptor for line segments that can be efficiently matched with a set of other line segment descriptors. They are to some extent rotation, illumination and scale invariant (the latter only if a scale space is used), and strongly dependent on sufficient structure in the area near the line segment.

2. Geometric constraints: Geometric and topological features are used to find matches. They can be based on spatial proximity, connectedness, angle constraints or whole graphs. The features are rotation, illumination and scale invariant. Matching is more expensive because there is no simple distance measure and the connected components must be taken into account.

3. Combination of local appearance and geometric constraints, also using point features.

The knowledge of global constraints like a stereo setup or continuous image sequence can be used to optimize the matching process by pre-filtering impossible matching pairs or reducing the searching space for possible matches.

9.2 Line Descriptors

To estimate a similarity between two line segments, first approaches have been made using normalized cross-correlations [193] or the sum of squared differences of the image information along the line segment or, as a more robust approach, whole patches along line segments (at the cost of more runtime). A major problem with these approaches to finding line segment correspondences is the additional degree of freedom along the line direction. Compared to point matching, an additional step is required. If a unique starting point is not known, the line segments have to be shifted on top of each other to find the optimal correspondence value.
To overcome this problem, a point-based approach can be used that searches for local point features near line segments [52]. In this case, additional effort is required to extract the point features, making the method a hybrid approach. While this approach fails when no texture information is available near lines, it can handle other problems quite well, since most operations can be performed efficiently on point-based data, but also provide topological line segment-based information.

Another solution is to use metrics that are independent of the localization of the line segment in the image and can be directly compared. This can be achieved with descriptor-like approaches. In the literature, different approaches such as histograms of neighboring color profiles [4] or mean and standard deviation of image gradients [226, 236] along line segments are used. Like point features, this information is stored in fixed-size descriptors for efficient comparison (e.g., by computing the norm between two descriptors).

9.2.1 Descriptor Composition

Similar to point feature descriptors, there are several variants for line segment descriptors. A basic concept was introduced in 2009 by Wang et al. [226] with the Mean Standard Deviation Line Descriptor (MSLD). Based on the gradient images in X- and Y-direction, the mean and standard deviation are computed along the line using a specified number of subregions M (stripes parallel to the line) and subregion size. The subregions are weighted similarly to the SIFT descriptor based on a Gaussian function, reducing the importance of subregions further away from the line. Additionally, the gradient samples of subregion i contribute to the two adjacent subregions $i+1$ and $i-1$ to avoid boundary effects. Since the gradient data is not rotation invariant, the gradient samples are aligned to the given line segment. The gradient samples below 0 are treated separately. Thus, each sub-region is composed of 8 values, the final descriptor size is $8M$.

Then, the mean and standard deviation parts are normalized separately to make them invariant to linear changes in illumination. Additionally, the influence of non-linear illumination changes can be reduced by limiting each value to a certain threshold (0.4 is proposed by MSLD). In this way, the importance of gradients with strong magnitude is weakened, while the importance of the orientation distribution is strengthened. Finally, the entire descriptor is renormalized.

A short time later, the same authors published an extended method called Harris based Line Descriptor (HLD) [225], which uses the Harris feature to achieve rotation invariance. Zhang et al. published a similar descriptor (based on the MLSD) called Line Band Descriptor LBD [236]. The main extension is the use of an image pyramid to integrate a scale space. To improve the performance for real-time applications, Hirose et al. introduced the Line-based Eight-directional Histogram Feature (LEHF) [90]. Instead of using the gradient information of the whole area around the line segment, only a sparse subset is used to compute the orientation histograms.

Depending on runtime requirements and descriptor distinctiveness, the introduced MSLD approach can be easily extended. To improve the distinctiveness, additional features can be included in the descriptor, such as color information or phase. Instead of simply sampling the feature data sparsely using a grid, a random approach could be used to obtain statistically more stable descriptors.

9.2.2 Left Right Descriptor

While the aforementioned hybrid approach of Fan et al. [52] considers the left and right side of a line segment, the MSLD approach including the mentioned extension misses this distinction. Creating a descriptor based on both sides of the line segment can result in well-distinguishable line segments being rejected in the matching

process. Often, edges within an image are occluding edges of foreground objects. Even small changes in viewpoint can cause large changes in the background side of a given edge. By creating two separate descriptors for the left and right sides of a line segment, this problem can be solved by simply using the smaller descriptor distance as similarity. However, for line segments in low texture areas, this may result in more false matches.

9.2.3 Orthogonal Descriptor

The fixed size descriptor methods discussed so far use the structure parallel to a line to create the descriptor. This approach has the advantage that the matching candidates can be computed quite efficiently. While for point features the location of the matching candidates is also uniquely determined, this is not the case for line segment features. The matching line segments can be shifted along themselves, making the location ambiguous.

Since in most cases structural information is available, which can help to solve the localization ambiguity of the matching candidates, an additional orthogonal descriptor is proposed. As a basic solution, whole image regions around two matching line segments could be extracted and then shifted in line direction until the minimum difference is found. Since whole image regions have to be extracted and transformed in order to compute the differences of the shifting patches, the optimization is quite expensive.

To create the fixed size parallel oriented line descriptor, the area around the line segment is analyzed and must be traced. At the same time, the orthogonal descriptor can be created with minimal overhead. For each orthogonal line along the line segment, the 8 values are computed based on the MSLD approach. The final descriptor size $ds = 8L$ depends on the length of the line segment L. Again, the left and right sides of the line segment should be considered separately, increasing the descriptor size by a factor of two.

9.2.4 Linked Descriptor

Based on the topological information given by a feature graph, as discussed in subsection 7.2.2, the edge or line feature descriptors can be optimized or augmented. For close parallel edges, lines could be detected and combined as a single descriptor to reduce the number of candidates for matching. Alternatively, the descriptors of parallel edges can be combined as a group to reduce ambiguity.

Another approach is to combine descriptors of connected line segments based on an edge segment. This allows curvy edge features consisting of many short line segments to be aggregated into a single descriptor or a group of descriptors. This is advantageous when using image information that is distorted, e.g. by skipping rectification or by using special lenses for the camera to extend the field of view. A similar approach has been introduced by MSLD. But instead of combining several line segment descriptors, they proposed to compute the descriptor for curved edge segments directly. This is done by using the gradient orientation to analyze the region around the edge segment to compute the mean and standard deviation value pairs.

9.2.5 Descriptor Matching

To find a match between two descriptors $\boldsymbol{a}$ and $\boldsymbol{b}$, a distance measure must be introduced. In general, the Euclidean norm of the difference vector resulting from the two descriptors is used: $d = \|\boldsymbol{a} - \boldsymbol{b}\|$. Other distance measures are possible, such as the sum of squared differences or the sum of absolute differences. The computed distance can then be used to evaluate the quality of the match, e.g. to remove match candidates by applying a threshold.

For a left-right descriptor pair, the norm of both sub-descriptors is computed and the smaller distance is used as the final distance. This has the effect of preserving more candidate matches. For descriptors based on occluding edges, this prevents them from being rejected if the background side of the edge strongly changes. But for less descriptive edges, this has the side effect that more false matches are found.

For orthogonal descriptors, the distance measure is computed in the same way as for normal descriptors. However, not all entries are used. Instead, the length of the smaller of the two descriptors is used to define the excerpt that is used to compute the distance. Then the smaller descriptor is shifted over the larger one and with each shift the distance is calculated. The smallest distance is used as the result. Additionally, the shift position for the smallest distance can be stored as an optimized localization of the two corresponding line segments.

Linked descriptors must compute a combined descriptor of all linked line segment descriptors. This can be done by summing the linked descriptor values and renormalizing the resulting descriptor. In addition to the resulting similarity of the linked descriptor pair, other characteristics such as the number of linked elements, etc. could be used.

If there are also orthogonal descriptors, they must be combined depending on the type of link. If a group of parallel line segments is given, the individual orthogonal entries can also be combined by summation and renormalization. In the other case, they can be combined into one large orthogonal descriptor by successively merging them.

9.3 Line Feature Matching

Feature matching is a complex process and includes several steps to obtain the final result. The steps involved may vary depending on the application problem. A common processing sequence for feature matching can be summarized as follows:

1. Compute the features of image A and image B and create a list of candidate matches.
2. If there are constraints between the two images, apply the constraints to reduce the number of detected features and match candidates.
3. Compute the descriptors of all remaining features and the distances of all remaining candidate matches by the descriptors.
4. Reduce the candidate matches by removing matches with large distance and apply other outlier detection methods, to receive stable results.

The constraints that exist between images depend on the application setup and the application problem. For example, in a fixed multi-camera setup, the epipolar geometry can be used as a constraint to find matches between the simultaneously captured images from the multiple cameras.

9.3.1 Matching Candidate Filtering by Constraints

As initial matching candidates, each detected feature of image A can correspond to each detected feature in image B. Let n be the number of features in image A and m the number of features in image B. Then the resulting number of candidate matches is $c = \frac{nm}{2}$. After computing the distance for each candidate and applying a threshold to remove bad matches, in most cases there will still be many false matches that need to be sorted out. This is a non-trivial and time-consuming task. To mitigate this problem, impossible matching candidates should be removed before the matching process.

For a given fixed camera setup, the epipolar geometry [83] can be used to filter out impossible match candidates. In the case of line segment features, this can significantly reduce the number of match candidates. Since only point constraints are defined by the epipolar geometry, the endpoints of a line segment must be used. By determining the epipolar lines in the second image of the two points given by the line segment of the first image, the matching candidates for the given line segment can be found, since all candidates must lie at least partially within the area spanned by the two epipolar lines [194].

A special case of epipolar geometry, also known as stereo setup, further simplifies the search. If two cameras are mounted with a baseline orthogonal to the viewing direction, the epipoles are placed in this direction at infinity, forcing the epipolar lines for a given point to be parallel to the moving direction. Additionally, a scale is given by the move distance, since all distances can be related to the move distance.

The epipolar geometry constraints can also be applied to a sequence of images from the same camera, if the location of the camera is known for the frames. This can be provided by GPS, IMU, optical flow, or predicted data. But even if the initial position is not known, a strong filtering of matching candidates can be realized. Depending on the frame rate, up to a certain motion or rotation speed, the movement of the features from frame to frame can be considered as small and within the same scale. By applying a filter based on local proximity, many candidates can be filtered.

Again, motion or rotation prediction can help to further restrict the search area. Many real-time image sequence approaches often do not use any descriptors at all for the tracking problem. The features are tracked directly using optical flow and/or multi-scale approaches [20]. For example, the ROVIO framework [137] uses patches for point features and estimates the position change of an entire image region. The same approach can be applied to patches along line segments.

Finding feature correspondences in a scene from arbitrary perspectives (wide baseline) is much more challenging. Scale, rotation, and illumination invariances must be taken into account. In contrast to common point descriptors like SIFT, scale invariance to line descriptors is not built into the descriptor. Approaches exist to simply compute the descriptors at different scales and include all of them in the

search for matching candidates [236]. If some position estimation is available, e.g. by GPS and orientation sensors, the epipolar geometry constraints can be used again. For weaker perspective changes, it can be tried to estimate a global image rotation [236] to remove impossible matching candidates.

9.3.2 Final Line Descriptor Matches

After determining the descriptors and computing the distances of the candidate matching features, a simple threshold is first applied to reduce the number of candidate matches that have sufficient similarity. Now, for each feature in image A, the matches of features in image B or vice versa can be found by applying a nearest neighbor search (KNN) [17]. If only one match is found in image B for a given feature in image A, it is accepted. If there are multiple matches for a given feature in image A, but the distance of the best match in image B is significantly smaller than the other distances, the match with the shortest distance is accepted. To ensure the decision of a match, a left-right check can be performed. In this case, the match list for the match candidate feature of image B is considered. If the initially used feature of image A is the only one or the first one (shortest distance) in the match candidate list of the given feature of image A, it is accepted.
Because line segments can be fragmented, multiple matches for a given feature may be correct. For example, if the match list for a given feature contains several matches with small distances and others with much larger distances, it is worthwhile to take a closer look at the first set of matches. The line segments corresponding to the (sub)list of matches can be tested for local proximity and orientation similarity. If they pass the test, multiple matches are accepted.

All other matches are considered ambiguous. To resolve the ambiguity, different strategies can be used to find the final matches. One approach is to analyze the regions of the found candidate matches in more detail. This can be done by using a patch-based comparison or by applying the orthogonal descriptor as discussed in subsection 9.2.3.

The second strategy is to use the topology of the line segments. Based on found unique matches, the ambiguity of adjacent or nearby features can be resolved by testing for similar topology [35]. Alternatively, subgraphs based on the topology can be used as unique patterns to be found within the topology of the features of the target image and uses subgraph matching approaches [112].
A third approach is to use global optimization methods. An example is the method introduced by Zhang et al. [236]. It creates a relational graph based on the candidate matches. The nodes represent the potential correspondences between two candidate matches, and the edges represent the pairwise consistency score between the nodes. The relational graph is represented by an adjacency matrix, which can be efficiently solved using the spectral technique introduced by Leordeanu et al. [122].

9.3.3 Outlier Detection

The resulting list of matches usually contains outliers. For stable post-processing, e.g. tracking, it is important to reduce or eliminate the outliers. This requires a global constraint between the two images that allows to estimate a quality measure of the matches. This can be done by using the epipolar geometry or in particular the fundamental matrix [139], also called the bifocal tensor. With the fundamental

matrix F the epipolar line $l = Fx$ for image B is given by the point x of image A. The corresponding point x' in image B must lie on the epipolar line l. The fundamental matrix can be computed from 8 or more point correspondences. By computing the epipolar lines for image B from the points in image A, the distances of the corresponding points in image B to the epipolar lines can be estimated and used as a quality measure for the point correspondences. This measure can be used by the RANSAC method [59] to find a sufficiently accurate fundamental matrix and to detect outliers.

This approach can only be applied to point matches. For lines, the epipolar geometry can only help reduce the search range for candidate matches, as discussed in subsection 9.3.1. A similar approach requires matches between three images. This is called the trifocal tensor [84]. If the camera parameters are known, a line on the image plane can be back-projected as a plane in 3D space. If three back-projected planes meet in a single 3D line, a true constraint on a corresponding line set is provided. Based on this constraint, the trifocal tensor can be determined as a 3x3x3 matrix and requires 7 point or 13 line correspondences between 3 images [212].

9.4 Results and Discussion

This chapter provides general information on line descriptions and line descriptor matching, and outlines key concepts and methods. However, the detailed results and discussion are omitted here to avoid overlap with Manuel Lange's work. His thesis already covers these aspects in detail, including a thorough analysis and evaluation of the results. For further details, please refer to his thesis [118].

Chapter 10

Line Extraction Framework

Most of the approaches and algorithms discussed in this thesis are implemented in C++ and provided by the Line Extraction Library. In addition, a comprehensive Analyzer application has been developed. It allows to experiment with the line extraction methods and provides rich visualization tools to analyze the results. In addition to providing numerous examples and case studies, the framework also includes evaluation tools to simplify the setup and comparison of different line extraction or edge response approaches.

At first glance, the provided implementation may appear to be a collection of libraries with additional tools rather than a framework. However, a separate section will explain how and why this implementation can be interpreted as a framework rather than just libraries. This discussion is preceded by an overview of the design and architecture of the framework and its associated tools. After a detailed analysis of the framework aspects, the third section introduces the Analyzer application, and the last section gives an overview of the evaluation tools.

The framework is open source and can be found on GitHub [227]. It is intended to allow developers and researchers to experiment with and evaluate different line extraction methods, and to provide a foundation for further research and development in line feature enhanced computer vision and image processing.

10.1 Line Extraction C++ Libraries

The libraries are designed to provide a modular and extensible architecture, allowing seamless integration of new features and optimizations. Implemented in C++, it leverages the language's performance advantages to support the development of fast and efficient computer vision algorithms. The implementation was initially based on version 11 of C++ (C++11), and is in the meanwhile upgraded to version 17 (C++17). This may change again in the future to take advantage of the latest features and optimizations that C++ then provides.

Most of the libraries are designed as a template library (header only), which allows flexible configuration at compile time in many cases, e.g. choosing the numerical precision for calculations (int, float, double, etc.) or generally stacking different types of methods for evaluation as described in section 10.2. By using C++17, it can take advantage of modern C++ features for improved code clarity and performance.

The build system of the library is primarily realized using Bazel [132], which provides hermetic, reproducible builds with automatic dependency management via the Bazel Central Registry (BCR). CMake [108] is maintained for legacy compatibility. Both

build systems allow to easily configure and build the library with various compilers and on different platforms. Thus, it is cross-platform enabled and can be compiled on Windows, Linux and MacOS (tested so far on Windows and Linux). The library is also designed to be easily integrated into multiple projects, as it is mainly header-only and has few dependencies.

10.1.1 Framework Dependencies

The core dependency of this framework is the OpenCV library [26], which offers a comprehensive set of tools and algorithms for image processing and computer vision. The library is used for tasks such as image loading, image manipulation, feature extraction and image output (display and file). Basic types such as cv::Mat, cv::Point, etc. can be found throughout the code base.

To address specific tasks, such as optimization techniques or general mathematical computations, additional libraries and tools are included:

- **eigen**: The eigen open source library [75] is a C++ linear algebra template library that provides a wide range of matrix and vector operations. It is mainly used in the library for matrix operations and numerical computations, especially in the geometry package for describing objects in 3D space.
- **dlib**: dlib [106] is a modern open source C++ toolkit containing machine learning algorithms and tools for creating complex software in C++. It is mainly used in the library for optimization tasks using the provided numerical algorithms.

Additional dependencies are used for tasks such as binding tools to Python, visualization, graphical user interface development or experiments outside the framework's main domain. However, they are not listed here, as they do not directly contribute to the discussion of line feature extraction methods and are generally not required for using the framework in other projects.

10.1.2 Project Structure

In addition to the main libraries, where the sources are located in the include and src directories, the project includes several other directories containing standard components such as documentation, build scripts, unit tests, and examples. It also provides the evaluation tool and the analyzer application:

- **libs**: Library sources, organized as sub-libraries (edge, lsd, lfd, etc.), each with its own `include/`, `src/`, and `tests/` subdirectories. See subsection 10.1.3 for details.
- **tools**: Contains build scripts, Bazel configurations, and development tools.
- **evaluation**: Contains thesis-specific evaluation code and performance benchmarks. The reusable evaluation framework (tasks, runners, metrics) is part of the `libs/eval` library, see section 10.4.
- **examples**: A collection of examples and case studies that demonstrate the usage of the library and its tools.
- **apps**: Currently it only contains the Analyzer application, see section 10.3.

10.1.3 Framework Structure

The core of the framework can be found in the `libs/` folder. Each C++ library has its own directory structure with `include/` (headers), `src/` (implementation) and `tests/` (unit tests) subdirectories. They provide different levels of functionality and are separated by their purpose and dependencies. The following list gives an overview of the available libraries:

- **utility**: General tools that are used within the other packages and provide basic functionality such as common interfaces, file system operations, automation tools, tracing and result logging, string manipulation, and basic computer vision operations such as interpolation.
- **eval**: Tools for evaluation like tasks, test runners, data providers, metrics and reporting.
- **geometry**: Tools for geometric operations, such as point, line, and polygon representations, as well as geometric transformations and operations.
- **imgproc**: Collection of image processing tools and methods. It mainly covers the edge response extraction algorithms as discussed in chapter 2 and chapter 4 like the Sobel operator or the Laplacian of Gaussian (LoG) operator.
- **edge**: Provides a collection of edge-based (edge response, edge pixel, edge pattern, etc.) algorithms. It includes methods from NMS to connect edge tracing to line fitting methods based on edges. It covers most of the methods discussed in chapter 7 and chapter 8, but also chapter 6.
- **lsd**: The heart of the framework. It provides a generic interface for line segment detectors and includes implementations of all the methods and approaches discussed in chapter 8.
- **lfd**: This package provides a collection of line feature detection, matching and filter algorithms. It covers the methods discussed in chapter 9.

The libraries and their associated tooling follow a defined software architecture, which is discussed in the next section.

10.2 Framework Design

The libraries are not only a collection of independent tools but also provide a structured framework for specific tasks. This is particularly visible in the evaluation pipeline section 10.4, where new methods and tools must follow predefined interfaces to ensure seamless integration. It also enables the easy integration of new methods for the analyzer application section 10.3.

While many algorithms and methods are implemented in loosely coupled classes for easy reuse, the framework also provides a set of interfaces and base classes that define the structure and behavior of specific components. The components itself are again loosely coupled to be used on a different kind of abstraction level. For those exists yet another layer of interfaces and base classes, which allows the components to be used in a combined final processing unit.

More concretely, the framework provides image processing units that, for a given configuration and input image, produce a specific set of results after processing. These units are then used in the analyzer application or, with a further level of abstraction regarding input image sets and aggregated results, in the evaluation pipeline.

Because of this approach, the implementation can be considered both, a collection of libraries and a framework. It establishes clear conventions that enables the extension of existing components while maintaining compatibility. This design ensures flexibility and scalability, making it easier to integrate new line extraction methods, but still allowing the developer to use whatever he needs to solve his problem [172, 230].

Before examining the overall architecture, the next section first introduces the basic building blocks of the framework.

10.2.1 Filter Architecture

This section provides an overview of the filter architecture used in the framework. The architecture is designed to be modular and extensible. It allows various filter types to be implemented, based on FilterI base class.

The following types are the fundamental building blocks of the filter framework:

- **cv::Mat**: OpenCV data structure for image processing. Stores image data, edge responses, and filter outputs.
- **ValueManager**: Base class providing a generic way to set and get configuration parameters, e.g. used in Analyzer application or evaluation tooling to configure filters.
- **FilterI<IT>**: Abstract class for all filters, supporting image processing, retrieving results, and defining intensity ranges.

Each derived filter class provides a unique set of functionalities that allows specific types of edge response extraction. Below is an outline of main categories:

- **GradientI<IT,MT>**: Provides methods for computing gradient magnitude, direction, derivatives, and normalization of thresholds.
- **LaplaceI<IT,LT>**: Focuses on Laplacian responses, including different thresholding and range calculations.
- **QuadratureI<IT,GT,MT,ET,DT>**: Extends LaplaceI by introducing even and odd responses, magnitude computations, and phase estimation.
- **PhaseCongruency(Laplace)I<ET>**: Implements phase congruency computation, often combined with QuadratureI to provide enhanced image analysis.

Each interface and class in this framework supports various template arguments, which provide flexibility in specifying data types and computation precision:

- **IT**: Defines the image type (e.g., 8-bit, 16-bit, etc.).
- **GT**: Specifies the gradient type, used in derivative computations (e.g., short, float, or double).

- **MT**: Defines the magnitude type (e.g., int, float, or double).
- **DT**: Specifies the direction type for orientation-based computations (e.g., float or double).
- **LT**: Defines the laplace type for Laplacian-based filters (e.g., int, float, or double).
- **ET**: Specifies the energy type used in phase congruency and quadrature filtering (e.g., int, float, or double).

Figure 10.1: Class diagram of the filter architecture.

The class hierarchy, as shown in Figure 10.1, provides multiple filter implementations based on the defined interfaces. These filters, called Edge Response Filter (ERF) enable different types of image processing techniques, including edge detection, noise reduction, and feature extraction (see chapter 2 and chapter 4 for more details about the methods):

- **Gradient Filters**: Sobel, Scharr, Prewitt, other gradient-based methods, SUSAN, RCMG, and an adapter for quadrature filters (one quadrature filter as 3 gradient filters).
- **Laplace Filters**: Standard Laplace, LoG (Laplacian of Gaussian), and OpenCV's Laplacian implementation.

- **Quadrature Filters**: QuadratureG2, QuadratureLGF, QuadratureS, and QuadratureSF.
- **Phase Congruency Filters**: PCLgf, PCMatlab, PCLSqf, and PCLSq.

10.2.2 Edge Tools

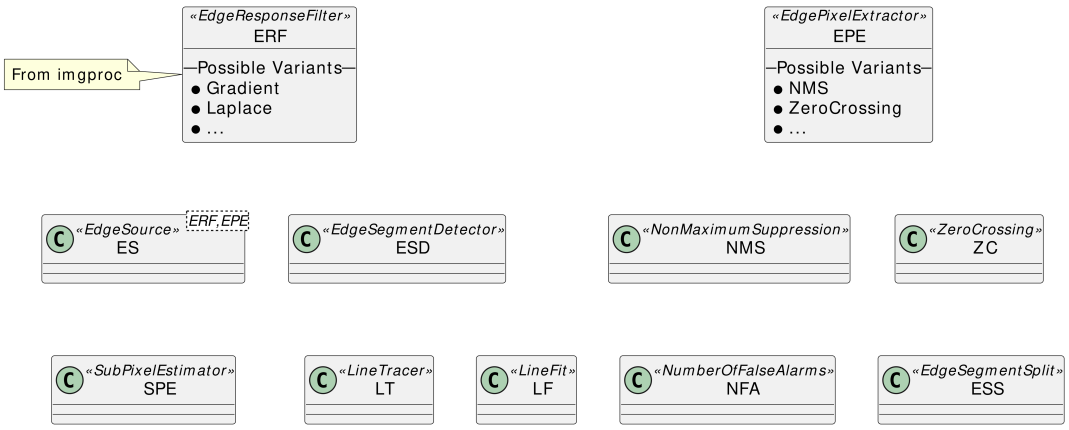


Figure 10.2: Overview of available edge tools.

This section introduces the edge tools provided in the edge src folder. Most of the tools found here are loosely coupled classes or C++ structs that implement a specific method or algorithm. In many cases there are different versions available, e.g. there are four different implementations for NMS. The main logic is implemented in separate C++ structs with the same "interface" (same set of functions), but all implementations are then combined in a base class by allowing them to be exchanged via a template argument. The same approach is used for most of the edge tools, which allows to easily extend new variants or algorithms with minimal effort to include them in the whole framework.
The following list gives a brief overview of the available tools (see also Figure 10.2):

- **Non-Maximum Suppression (NMS)**: Provides different implementations of the non-maximum suppression algorithm, including a precise calculation variant using a precise orientation map, and fast variants, using a 4 or 8 region orientation map (see subsection 6.2.2).
- **Zero Crossing (ZC)**: Similar to NMS, there are several variants, providing a precise version and two fast versions (see subsection 6.2.3).
- **Threshold**: Implements the threshold methods (including the Otsu method) as discussed in section 6.1.
- **Subpixel Estimator (SPE)**: Implements a subpixel precision estimator (see section 6.3 and subsection 6.3.1). Estimator (linear, quadratic, CoG, Sobel) and gradient interpolation (nearest, linear, cubic) are separated and interchangeable.
- **Edge Source (ES)**: Edge source interface (EdgeSourceI) and generic implementation that provides edge map, gradient map, etc. based on an edge response filter (see previous section) and an edge pixel extractor like NMS (see Figure 10.3 for an overview).

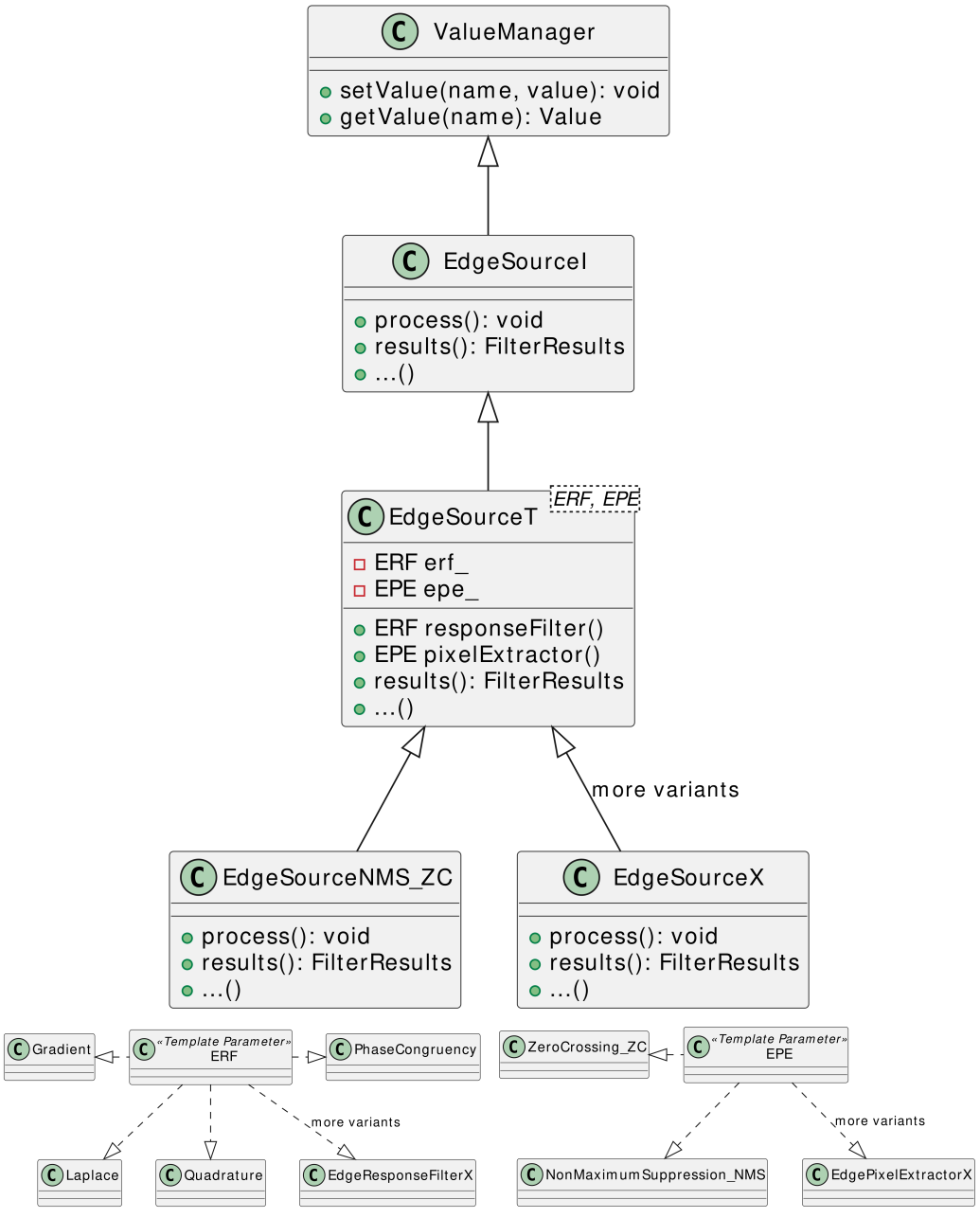


FIGURE 10.3: Edge source inheritance hierarchy. The lower part illustrates the template parameter options for ERF and EPE.

- **Edge Segment Detector (ESD)**: Provides a base class for edge segment detectors (EsdBase) that can be used to detect edge segments in an edge map. Variants for edge drawing, linking and pattern linking, as described in chapter 7, are implemented (also see Figure 10.4).
- **Edge Segment Split (ESS)**: Tool to split (large) non-straight edge segments into smaller straight segments. This is especially necessary for edge linking methods (see section 8.3) to get meaningful results.
- **Line Tracer (LT)**: Tool for extracting line segments from a given line, supported by edge points provided by an edge map. For example, the lines may come from a RANSAC or Hough transform calculation (see section 8.1).
- **NFA**: Implements the NFA (Number of False Alarms) calculation for edge segments (see section 7.4).
- **Line Fit (LF)**: Provides a base class for algorithms that fit line segments to a list of support (edge) points. This includes linear regression, PCA and M-Estimators (see section 8.4).

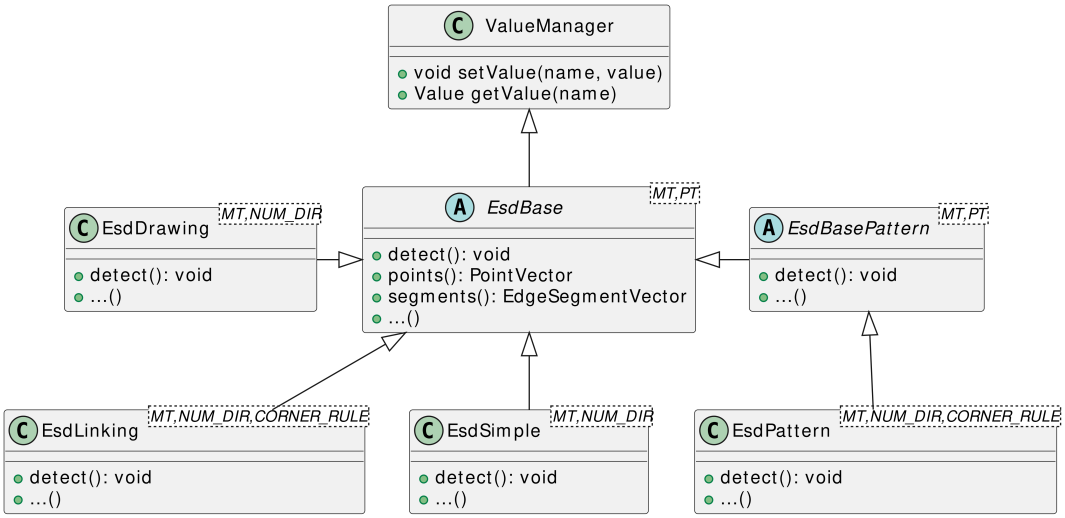


Figure 10.4: Edge segment inheritance and implementation hierarchy.

There are only two object class hierarchies in the edge tools, the edge segment detector (ESD, see Figure 10.4) and the edge source (ES, see Figure 10.3). The ESD is an integral part of the most advanced line segment detectors (LSD), which will be introduced in the next section.
Additional template arguments are:

- **CORNER_RULE**: Defines a non-typed template parameter that represents a compile-time flag to enable or disable the corner rule.
- **FT**: Defines the float type used for internal calculations. This template argument can be used to control precision and runtime behavior.
- **LPT**: Defines the line point type that refers to a point on a line segment. In general, this is a float type (FT) based 2D vector.

- **NUM_DIR**: Specifies a non-typed template parameter defining the number of directions to store in the edge map. Valid values are 4 and 8. This is a compile-time configuration option for the class.
- **PT**: Defines the point type that refers to a point in the datamap. Usually this is just a one-dimensional index of type size_t, but it can also be an integer-based 2D vector.

10.2.3 Line Segment Detector Architecture

The Line Segment Detector class hierarchy starts with the ValueManager, followed by the LdBase (line detector) class (see Figure 10.5). It provides the interface to detect lines in images. For line segments, there is a base class LsdBase, which is derived from LdBase. The distinction was made because methods like Hough transform or RANSAC (see section 8.1) do not provide line segments but lines.

However, there is an adapter that converts a line detector into a line segment detector using the line tracer tool. This allows the use of line detectors in the evaluation pipeline or other tools that expect line segments as output.

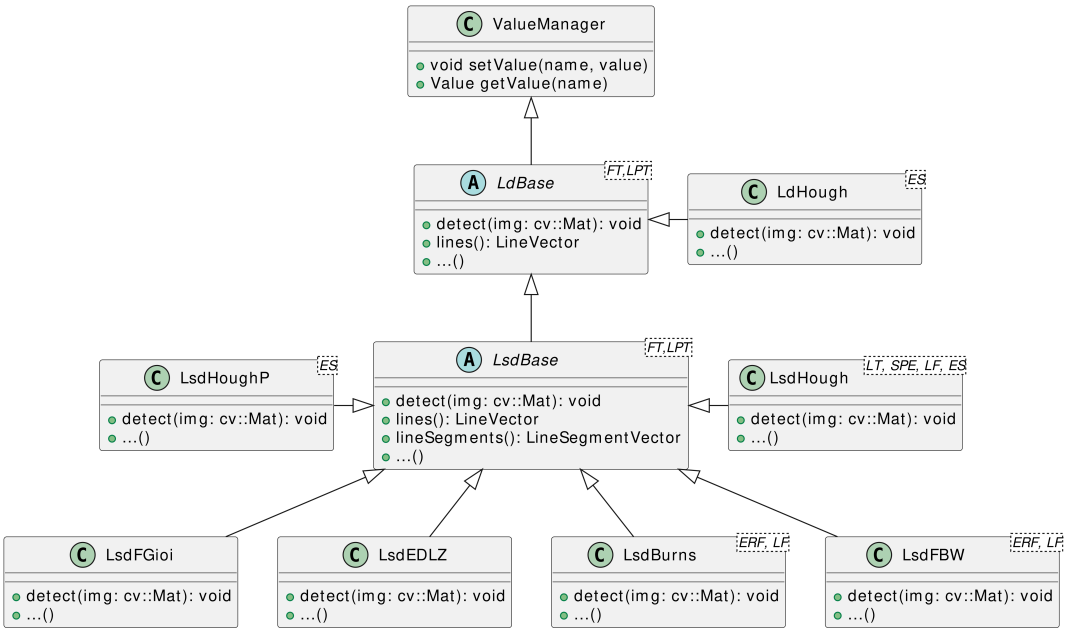


FIGURE 10.5: Line segment inheritance and implementation hierarchy down to LsdBase. The classes in the figure contain some template parameters that are not described in the two template argument lists. However, they refer to the edge tool abbreviations listed in the previous section.

There is only LdHough as LD implementation, together with LsdHough that uses the LSD adapter. Then there are LSD implementations based on known methods like LsdFGioi [67] or LsdEDLZ [3] (as described in section 8.2). They are based on the authors' original sources, but assimilated to the LSD interface. Except for the interface, they do not use much of the other available tools. However, they don't provide much internal data, e.g. for later analysis of the processing.

Other LSD implementations that also build on the edge and filter tools are methods like the probabilistic Hough transform based on opencv (LsdHoughP) or the reimplemented LsdBurns [29] method, or the custom detector LsdFBW. These methods reuse the existing edge extraction tools and build their own line segment extraction on top. The obvious advantage is that they can easily replace the edge detector and provide all processed data like edge maps, orientation maps, etc. for these steps.

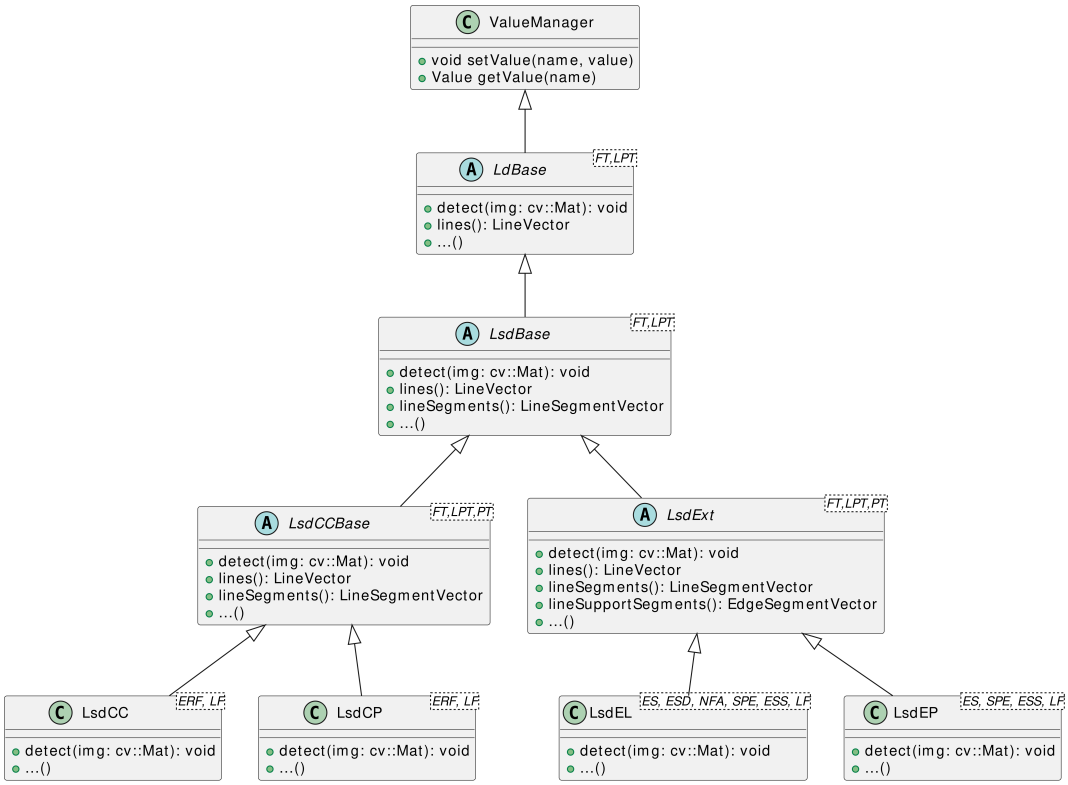


Figure 10.6: Line segment inheritance and implementation hierarchy of the extended interfaces LsdCCBase and LsdExt. The classes in the figure contain some template parameters that are not described in the two template argument lists. However, they refer to the edge tool abbreviations listed in the previous section.

Furthermore, there is the LsdCC (connected component) base class, which is used for the LsdCC and LsdCP (connected pattern) methods. These are fast and simple line segment detectors based on the connected component approach as described in section 7.2. Line segments are extracted using an edge segment fitting method that can be selected from the edge tools. The final line segments are not optimized or filtered, but extracted directly from the edge segments.

The last full line segment detector methods are the LsdEL (edge linking) and LsdEP (edge pattern) implementations. Like LsdCC and LsdCP, they are based on the edge linking approach, but also use the full range of available edge tools to extract the line segments based on the methods described in section 8.3 together with optimizations such as the (sub) pixel estimator or validation by one of the Number of False Alarms (NFA) methods.

Both classes inherit from LsdExt (see Figure 10.6), an extended interface that provides advanced access to processing data for analysis. They allow the selection of different methods for edge detection, non-maximum suppression, (sub) pixel estimation, edge segment splitting and edge-based line segment fitting. In addition, LsdEL incorporates multiple linking strategies by replacing the edge segment detector and allows evaluation using NFA methods.

The LsdEL and LsdEP classes are the most advanced line segment detectors in the framework. They provide a wide range of configuration options to enable or switch between most of the methods discussed in this thesis. Additionally, all data from each processing step that generates the line segments is accessible. This allows for in-depth analysis and evaluation of the data. Despite their high flexibility, they achieve very good performance because many configuration parameters are determined at compile time, allowing efficient compiler optimizations.

Several variants of these are available in the Analyzer application, which is introduced in the next section.

10.3 Analyzer Application

The Analyzer application is a tool with a rich graphical user interface (GUI) that allows a user to interact with most of the methods provided by the line extraction library. In particular it lets the user select an image and a line segment detector, configure almost all possible variables and visualize the results. It also enables the user to interact with the line segment results and apply additional optimization methods. The application is implemented in C++ and uses the Qt framework [207] for GUI creation and rendering. This allows it to be compiled and executed on multiple platforms like Windows, Linux and MacOS.

Since the application provides a lot of options and tools, this section only gives a rough overview of the available features. For more information, please refer to the documentation and the readme file in the [227] repository.

10.3.1 Overview

Figure 10.7 is divided into several regions that provide different options to control the Analyzer application:

1. The options in this region allow the user to select an image and a line segment detector. Via the "Options" button, the first region also provides a dialog to select some image processing options that will be applied when the image is loaded via the load button (see Figure 10.8 (A)). The Line Segment Detector drop-down menu (see Figure 10.8 (C)) allows to select a processing method. When the image is loaded using the "Load" button, the line segment "Process" button is enabled.

2. It provides a dialog to configure the selected line segment detector. The configuration is done in a table view. Each column represents a registered ValueManager value as an option. The user can select and modify the cells to adjust the detector options. The table header also provides a description for

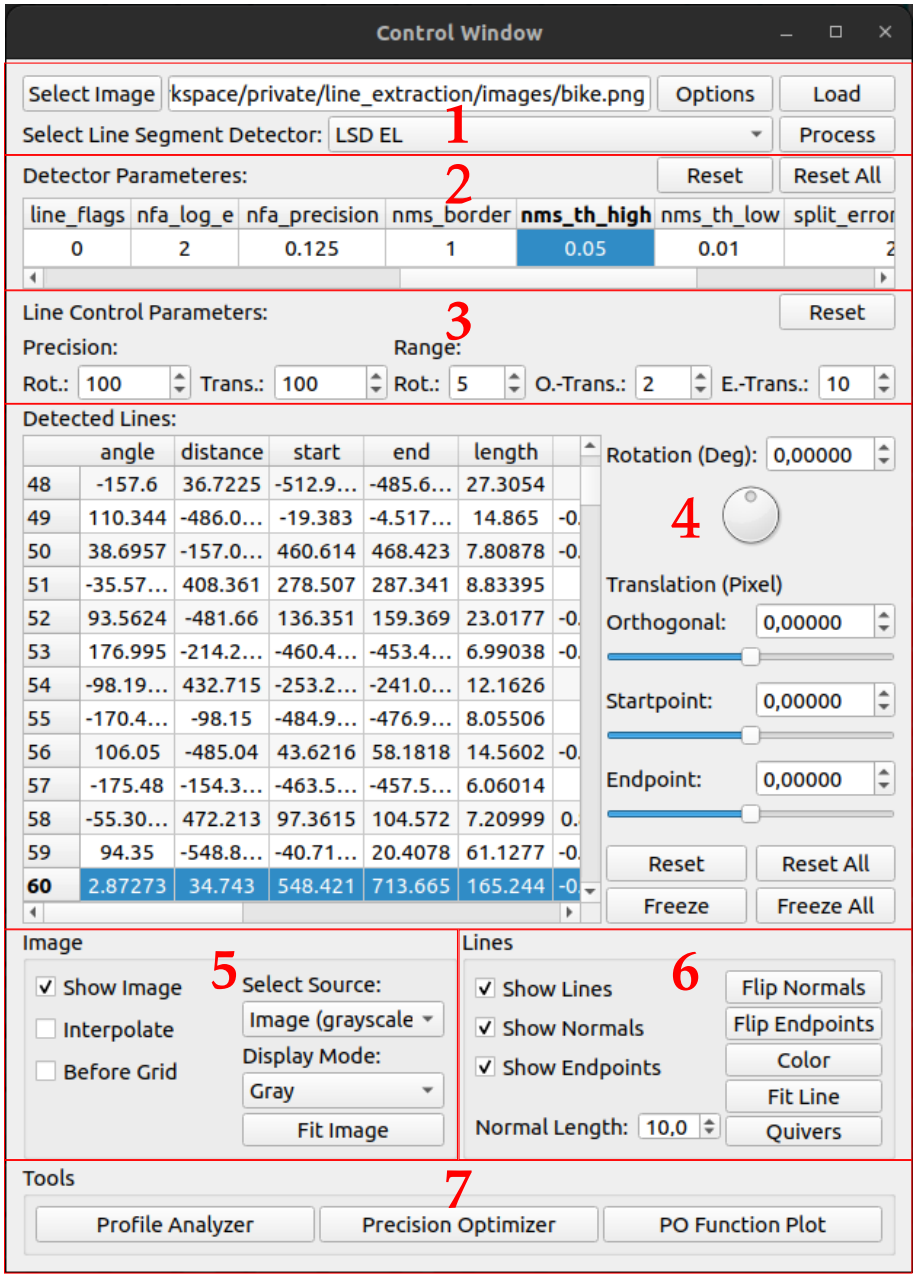


FIGURE 10.7: Screenshot of the Analyzer application showing multiple regions with different tasks: 1) Select image and line segment detector along with load options. 2) Line segment detector configuration. 3) Line control setup. 4) Left: list of detected line segments; Right: Controls to modify the selected line segment. 5) Image visualization options. 6) Line segment visualization options. 7) Available tools.

each option as a mouse over effect. The "Reset" and "Reset All" buttons allow the user to reset the selected or all options to default.

3. This region shows the line control setup. It allows the user to define the step precision for the rotation and translation steps that can be performed with the controls (see Figure 10.7 region 4 on the right). On the right side the valid ranges for rotation, max. orthogonal line translation and max. line segment endpoint translation are shown. The "Reset" button resets the line control parameters to their original values.

4. It shows the list of detected line segments. The user can select a line segment from the list and modify it using the controls on the right side or by directly modifying the value in the cell by double-clicking. The controls allow the user to modify the line segment by rotating (around the center of the line) and orthogonally translating (in the normal direction) it, and by translating the end points along the segment line. The "Reset" button resets the selected line segment to its original values, while the "Freeze" button sets the new values as default (overwrites the original values). The "All" buttons apply the corresponding function to all line segments.

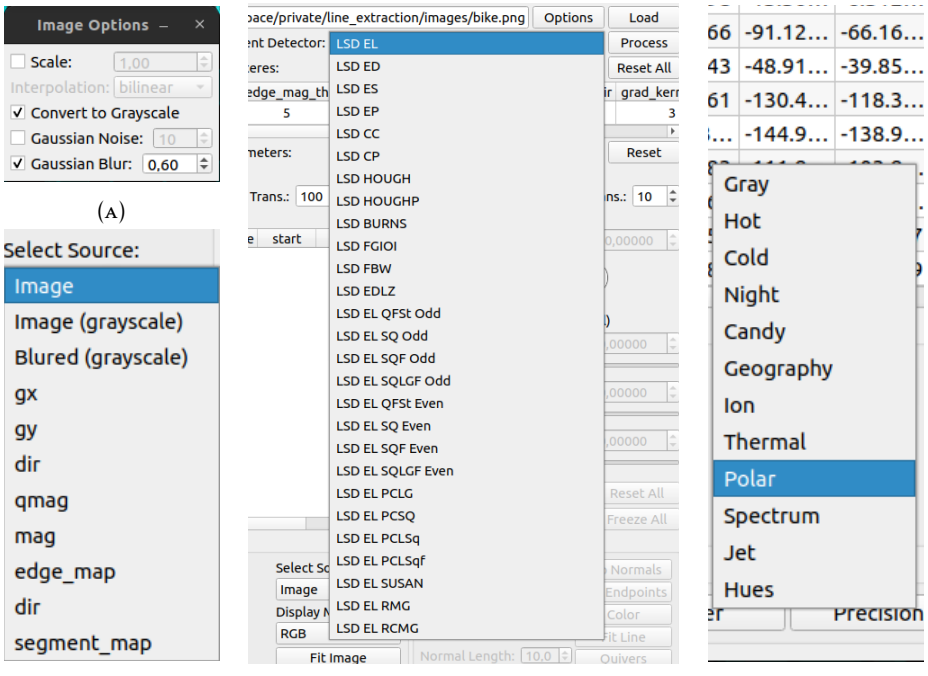
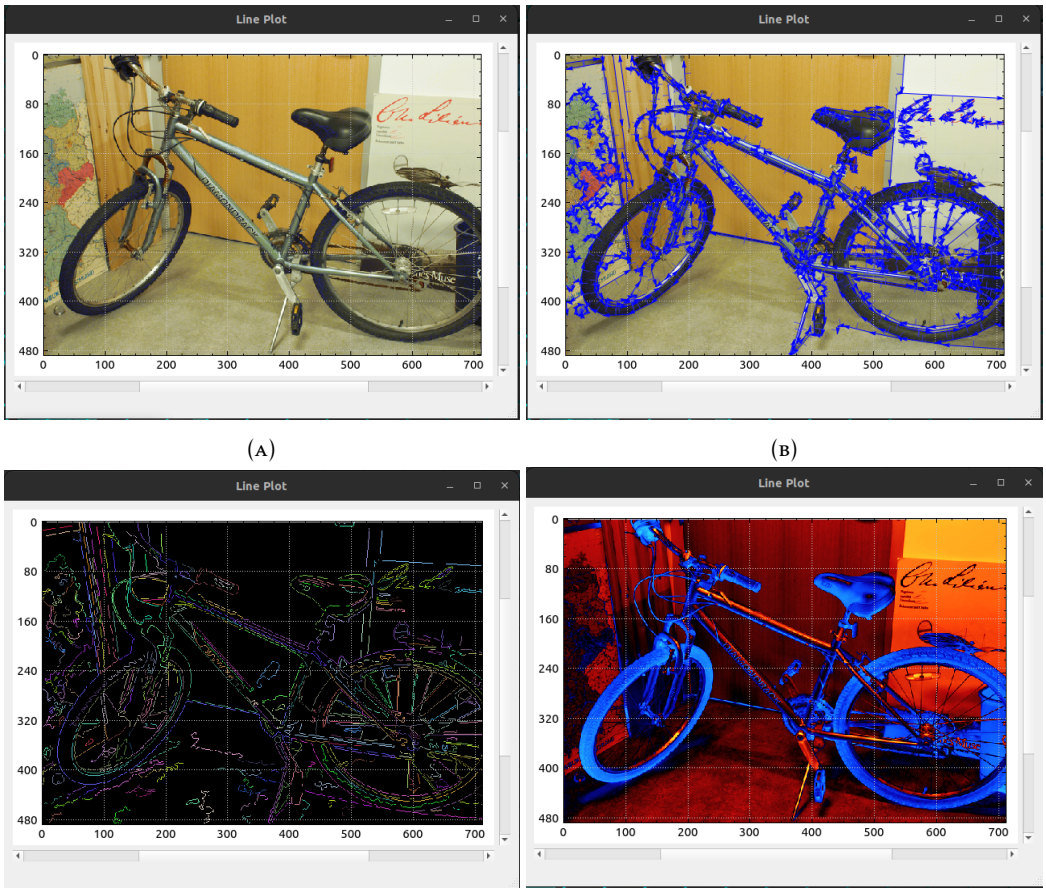


FIGURE 10.8: Options and dialogs: (A): Image load option dialog. (B): Image source selector for viewer. (C): Line segment detector method selector for processing. (D): Image visualization selector.

5. It provides image output controls. The user can select the image source (see Figure 10.8 (B) and Figure 10.9 (C)) and the visualization mode (see Figure 10.8 (D) and Figure 10.9 (D)). The "Fit Image" button automatically fits the loaded image to the current view size (see Figure 10.9 (D)). The options on the left allow to hide the image (e.g. if only the line segments should be shown), to improve the quality when zooming in by activating interpolation, and to move the grid of the view to the background.

6. This region provides controls for visualizing the line segments. The options on the left allow the user to hide the line segments or additional information such as the normal or end points. The buttons on the right allow the user to flip the line segment normal or end points, select the line segment color for visualization, and the "Fit Lines" button automatically fits a selected line segment to the current view size. The "Quiver" button provides additional tools for visualizing image data and is explained in subsection 10.3.4.

7. The last region provides a list of available tools. The "Line Profile Analyzer" and "Precision Optimizer" tools, along with the "PO Function Plot" tool, are explained in the following sections.



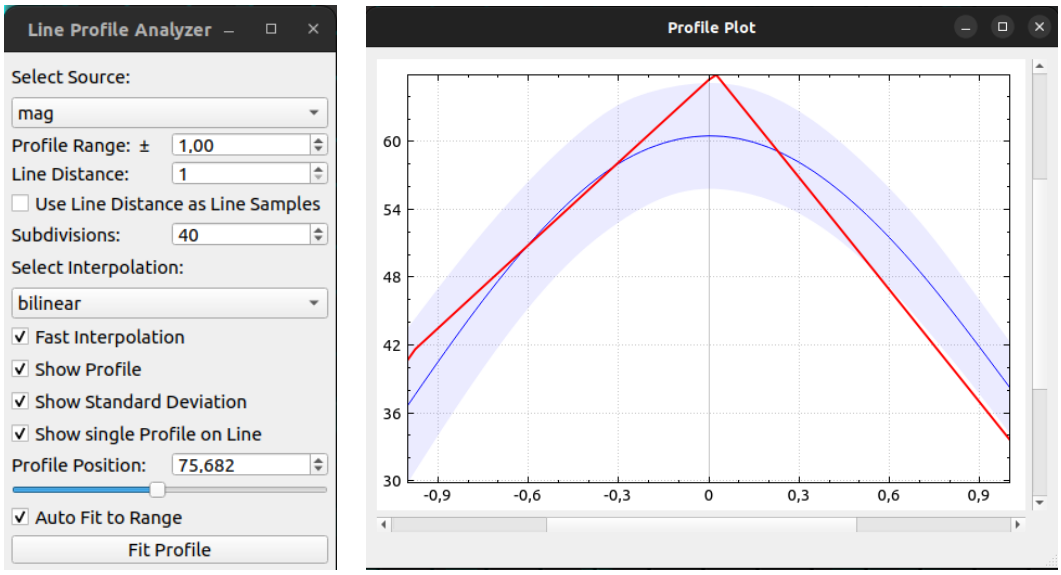
(A) (B)

(C) (D)

FIGURE 10.9: Analyzer Viewer to visualize images and line segments: (A): Loaded image; (B): Processed image showing the detected line segments; (C): Image source of edge segment map; (D): Image with polar visualization mode.

10.3.2 Line Profile Analyzer

The Profile Analyzer is a tool that allows the user to analyze the region around a line segment. It visualizes the average (see blue line in (B) of Figure 10.10) and standard deviation (light blue border along blue line in (B) of Figure 10.10) of the data source along a selected line segment by a given profile range. By also specifying a profile position, the profile information (orthogonal region to the line segment, see green



(A) (B)

FIGURE 10.10: Profile Analyzer Dialog (A) and Visualizer (B).

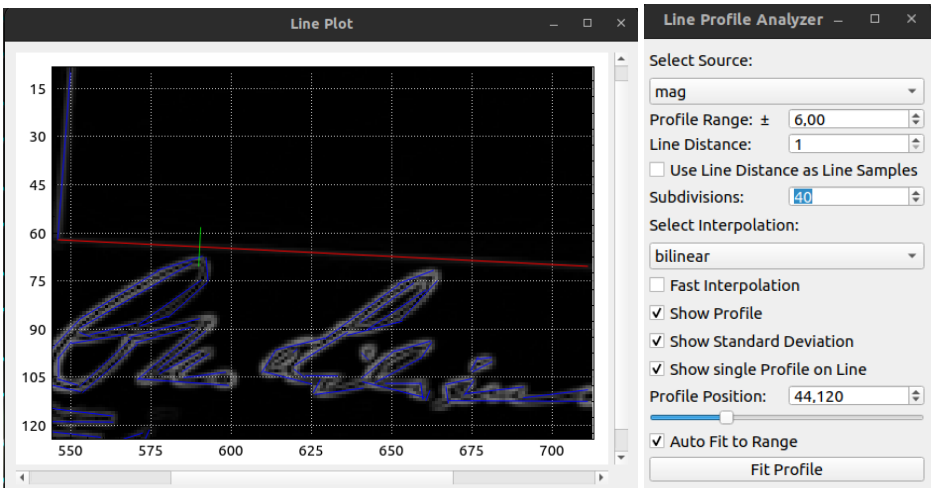
line in (A) of Figure 10.11) is plotted as a red line (see (B) of Figure 10.10 or (C) of Figure 10.11).

The dialog offers several options:

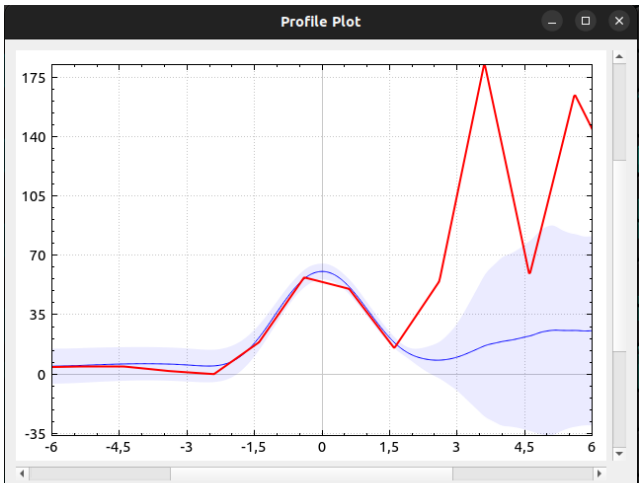
- **Select Source**: Enable the user to select a data source for processing (including all image data sources, see (B) in Figure 10.8).
- **Profile Range**: Allows the user to configure range that is analyzed (see (A) in Figure 10.10) and visualized. For an example with a larger range around the line segment, see Figure 10.11.
- **Line Distance**: This value allows the user to configure the sampling density along the line segment (1 means one sample for every response value along the line, 2 for every 2nd, 3 for every 3rd, etc.).
- **Use Line Distance as Line Samples**: This option changes the meaning of "Line Distance". Instead of directly specifying the sampling density by specifying the increment for sampling the response values along the line segment, the line distance is used to define the number of samples along the line segments, which also specifies the increment:

$$step\_width = \frac{line\_length}{line\_samples} \tag{10.1}$$

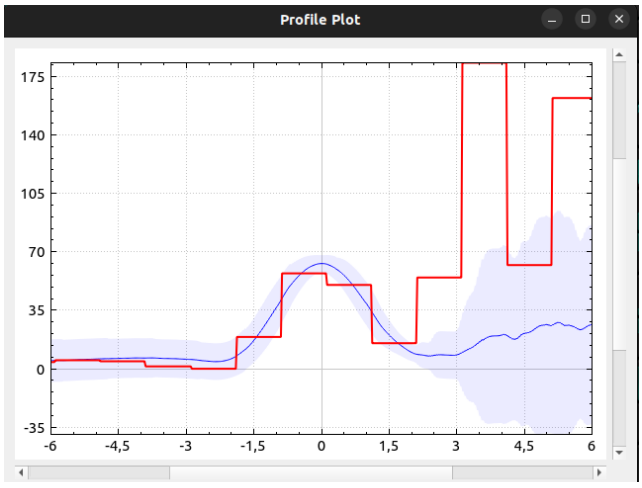
- **Subdivisions**: This option allows the user to specify the sampling rate for the curve plot (subdivisions = samples per range $[0,1]$).
- **Select Interpolation**: This dropdown allows to select one of several interpolation methods used to estimate the value for a given position (x,y) on the source plane from the index based source map (nearest, bilinear, bicubic) (see (C) vs. (D) in Figure 10.11).



(A) (B)



(C)



(D)

FIGURE 10.11: Profile Analyzer Example. (A) Visualization of the line segment and profile range as an overlay on the magnitude response image. (B) Settings dialog. (C) Visualization of mean and standard deviation (blue) of the data source along the selected line segment and in the current profile at the selected profile position (red). (D) Nearest (round) interpolation, showing the "pixel" ranges for each magnitude value in the response image.

- **Fast Interpolation**: Enables a faster but less precise interpolation option.
- **Show Profile**: Show average profile line in plot (blue).
- **Show Standard Deviation**: Show standard deviation regions in plot (light blue).
- **Show single line Profile on line**: Allows the user to plot a specific profile from a given position as a red line.
- **Profile Position**: Position on the selected line segment from which to plot the specific profile.
- **Auto Fit to Range**: If the range for the y-axis of the plot changes, the axis is rescaled to automatically fit the plot to the full height of the window.
- **Fit Profile**: Fit the profile plot to the current window size.

The example in Figure 10.11 shows a line profile with a rather large profile range. The supporting magnitude response for the line segment is nicely represented by a normal distribution-like curve with its maxima near the center of the range. While the left side of the curve smoothly approaches 0, the right side seems to have some noise in the average. Since the range is quite large (see the green line in (A) of Figure 10.11), it also collects the magnitude of unrelated responses, causing these irregularities. It is also represented by the red line in (C) and (D) of Figure 10.11, which directly represents the magnitude responses along the green line orthogonal to the selected line segment at the specific position selected in the Line Profile Analyzer dialog. The difference in the plot of the red line between (C) and (D) of Figure 10.11 is the interpolation method. In (C) it is linearly interpolated, while in (D) it uses the nearest magnitude value in the response map. In this case, it is more or less a visualization of the magnitude response profile cut orthogonal to the line segment at the given position along the line segment.

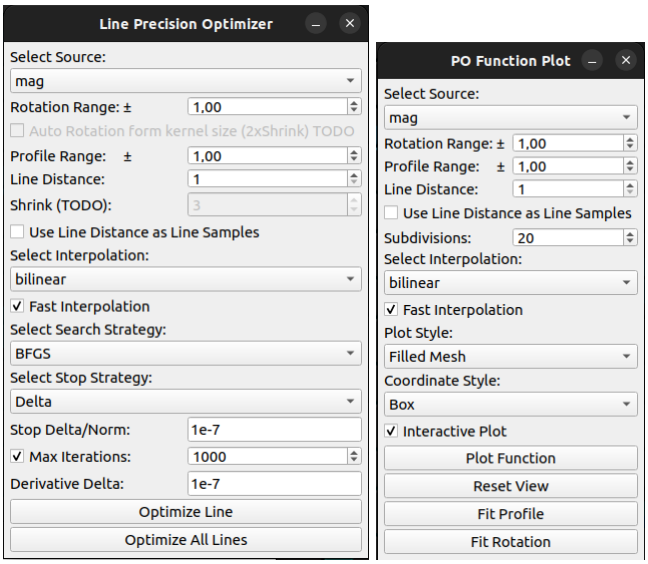
10.3.3 Precision Optimizer

The Precision Optimizer tool (see Figure 10.12) allows the user to optimize a selected or all line segments for an orthogonal translation or distance d and orientation (rotation around the center of the line segment) θ as described in section 8.6. With the PO Function Plot tool the error function surface can be plotted (see Equation 8.29).

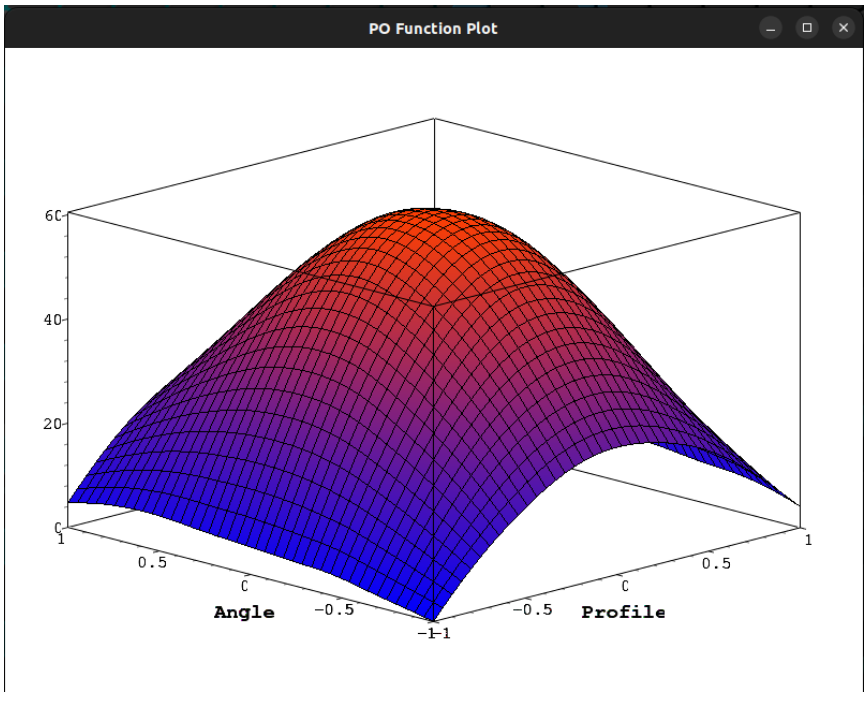
Both tools provide a set of similar options:

- **Select Source**: Select magnitude or energy source.
- **Rotation Range**: Rotation freedom in degree, for the line segment or the range that specifies the limits for θ: $[-rotation\_limit, rotation\_limit]$.
- **Profile Range**: The limit of the orthogonal translation or distance d, defining the search range $[-d_{\text{lim}}, d_{\text{lim}}]$.
- **Line Distance & Use Line Distance as Line Samples**: As described in the previous section.
- **Select Interpolation**: As described in the previous section.

The Line Precision Optimizer dialog additionally provides:



(A) (B)



(C)

FIGURE 10.12: Precision Optimizer and PO Function Plot tools. (A) Precision Optimizer dialog. (B) PO Function Plot dialog. (C) PO Function Plot visualization showing the average magnitude calculated from the support region of the selected line segment related to the rotation angle ("Angle" axis, degree) regarding line segment center and orthogonal translation ("Profile" axis).

- **Select Search Strategy**: Allows the user to select one of the available searching strategies:
  1. **BFGS**: Broyden-Fletcher-Goldfarb-Shanno method
  2. **L-BFGS**: Limited-memory-BFGS method
  3. **GCG**: Conjugate Gradient (optimized gradient descent) method
- **Select Stop Strategy**: Provides two stop strategies:
  1. **Delta**: The delta stop strategy is based on the delta from one iteration to the next. If the change is below a user-defined threshold, the search stops.
  2. **Grad_Norm**: This strategy looks at the norm (i.e. length) of the current gradient vector and stops the search if it is less than a user-defined threshold.
- **Stop Delta/Norm**: Defines the stop threshold for the stop strategy.
- **Max Iterations**: Additional stop criteria that limits the maximum number of optimization steps. Can be disabled.
- **Derivative Delta**: Delta used to compute the approximate derivatives for the optimization method.

The two buttons at the bottom of the dialog allow the user to apply the optimization to the currently selected line segment or to all line segments. The optimization is applied by changing the rotation and orthogonal translation values for each line segment in the control window. This allows the user to check and further modify the changes. To fully apply the optimization, the control can be frozen by pressing the "Freeze" button. Alternatively, the "Reset" button can be used to undo the changes (see 4 in Figure 10.7).

The PO Function Plot dialog also contains additional options:

- **Subdivisions**: The value is used for the same purpose as for the Line Profile Analyzer. The only difference is that the subdivisions are applied to both axes ("Angle" and "Profile") instead of just one.
- **Plot Style**: Let the user choose between different visualization modes (Wireframe, Hidden Line, Filled, Filled Mesh and Points).
- **Coordinate Style**: Specifies if and how the axes are drawn for the 3D plot.
- **Max Iterations**: Additional stop criteria that limits the maximum number of optimization steps. Can be disabled.
- **Derivative Delta**: Delta used to compute the approximate derivatives for the optimization method.
- **Interactive Plot**: Automatically updates the plot with every user change. Depending on the setting, the update can take quite a long time, so it is possible to disable it.
- **Plot Function**: User action to explicitly generate the plot.
- **Reset View**: Set the internal values (scale, position and rotation) of the 3d plot view back to its initial state.

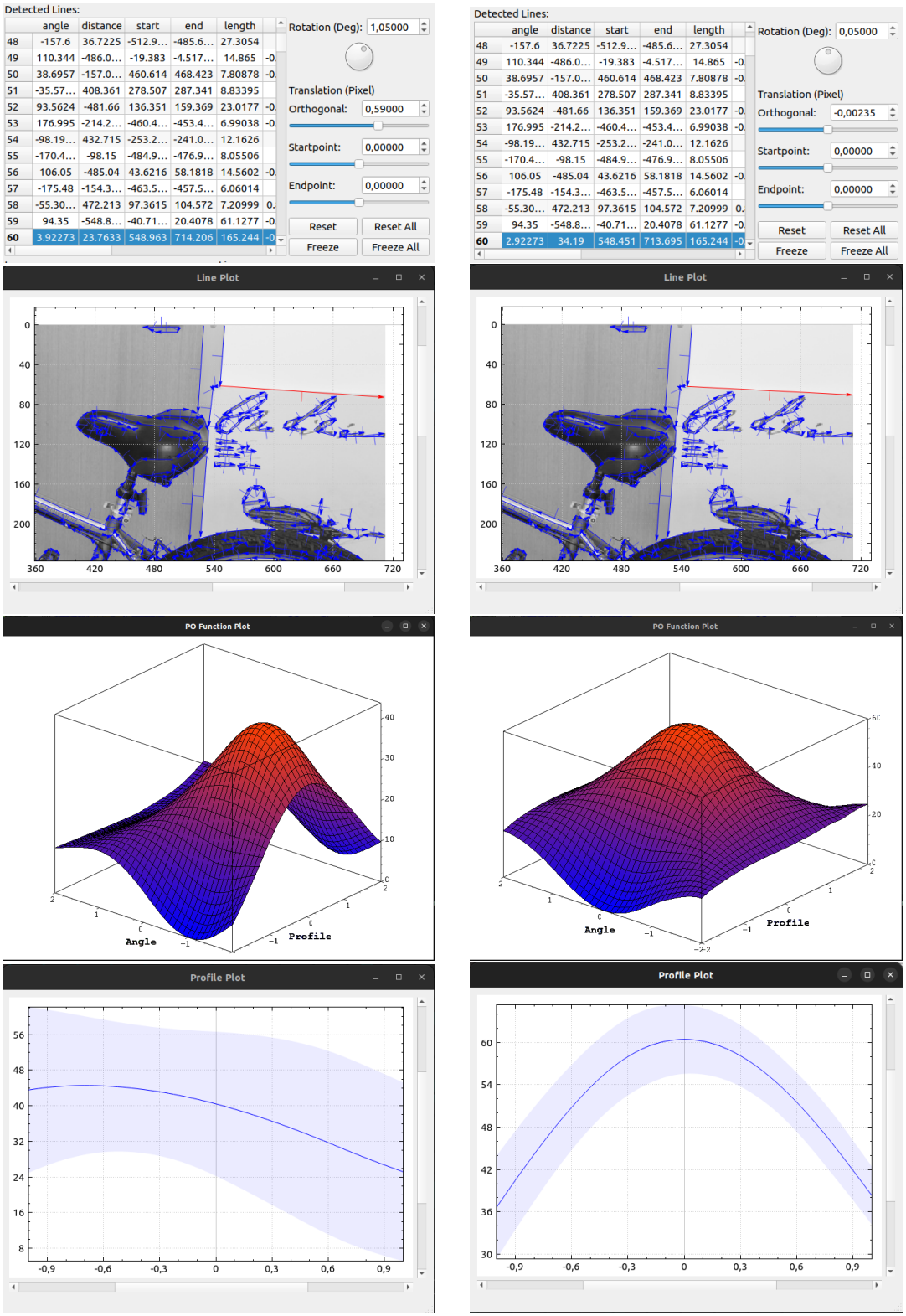


FIGURE 10.13: Optimization example: Left: Shows a line segment that has been manipulated by the user and is no longer perfectly aligned with the edge. Right: Shows the optimized, perfectly aligned line segment.

- **Fit Profile**: Aligns view to the "Profile" axis.
- **Fit Rotation**: Aligns view to the "Rotation" axis.

The optimization example in Figure 10.13 shows a user manipulated line segment on the left. It is slightly rotated and translated. The 3D plot on the left nicely visualizes the shift of the angle and also the profile. Neither is centered. In the 2D profile plot below, the profile shift is also nicely shown as the plotted line is shifted significantly to the left. The right side of the figure shows the optimized line segment. The optimization method used the "Angle"-"Profile" surface function from the 3D plot to optimize the rotation and translation toward the local maximum. Now both axes are perfectly centered on the right 3D plot. The 2D profile plot on the right is also much better centered, but not necessarily at the local maximum.

The PO Function Plot tool can also be used to visualize large-range 3D surface plots. This can be useful for investigating the proximity of local maxima. The example shown in Figure 10.14 shows that the local maximum of the selected line segment is quite isolated, so there is no chance of hitting the wrong maximum with the optimization method (as long as the starting point is still somewhere on the elevation caused by the local maximum). Far to the right, strong responses can be observed as the segment is moved toward the lettering in the image, which provides strong and dense edge responses. In this case, there are many local maxima with high proximity. This is due to the fact that the long line segment intersects with many edge responses in this area. If there were a weighting, e.g. by orientation proximity to the line segment, the elevation magnitude of the surface plot would be much smaller in this area.

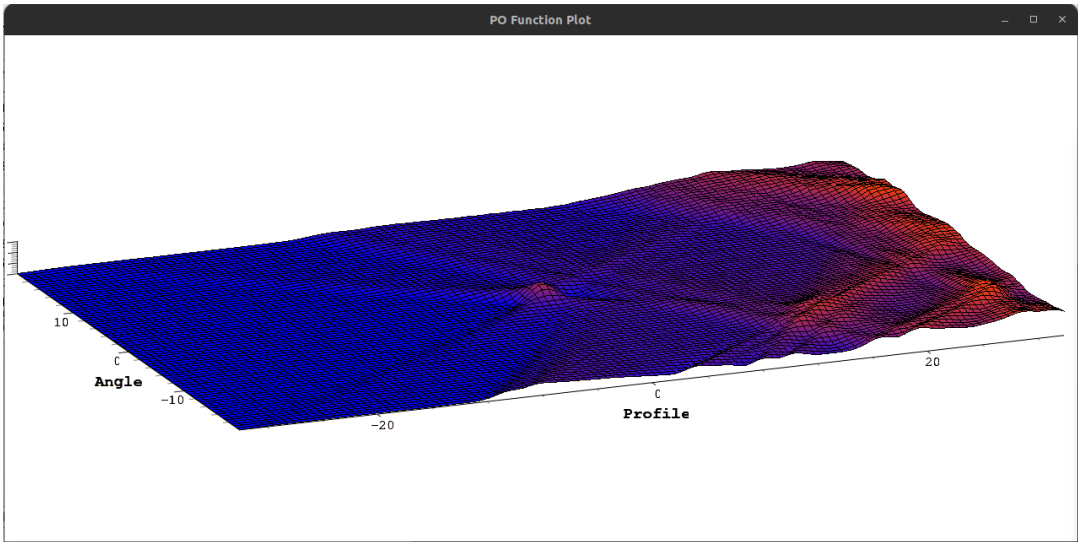
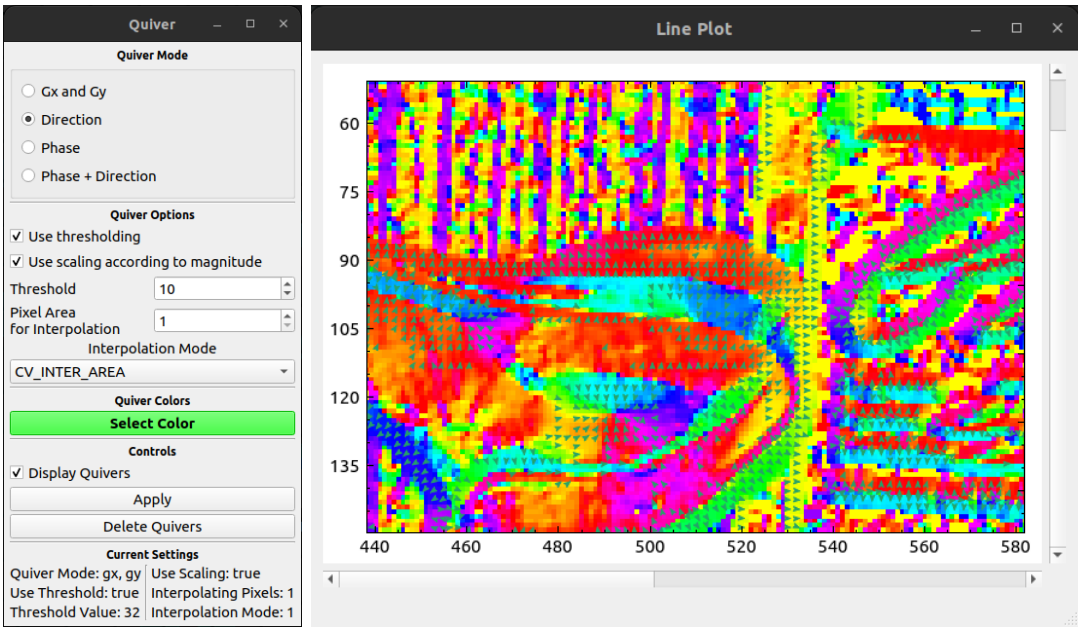


Figure 10.14: Large range 3D plot example. This is the same local maximum as in the other 3D plots. But in this case a rotation range of $[-20, 20]$ and a profile range of $[-30, 30]$ was configured.

10.3.4 Quiver Tool

The Quiver tool allows the user to visualize the response strength and direction of an edge pixel or area by filling the view with oriented and scaled small arrows. The orientation represents the level line (see section 8.2) or the orthogonal direction of the



(A) (B)

FIGURE 10.15: Quiver tool dialog (A) and example output (B).

edge area, while the length represents the response strength (see (B) in Figure 10.15). By defining a threshold for the minimum response and specifying an area of pixels from which the direction and strength are computed, the sparsity and density of the quiver representation can be controlled.

The tool also allows the user to select different types of input data to compute the quivers. However, the available input depends on the selected line segmentation method used to process the data. For example, the Phase mode can only be used if phase response data is available. The two options "Use thresholding" and "Use scaling by magnitude" allow the user to disable the sparsity filter by thresholding or scaling by magnitude. The tool also allows the user to select different interpolation methods that are applied to calculate the orientation and magnitude data based on the defined pixel area size. An option to select the color of the quiver is also available.

10.3.5 Customization and Extension

The Analyzer application provides a simple set of tools to extend the application with additional line segment detectors and analysis tools. To add a new detector, simply wrap it in one of the detector abstractions which, depending on the detector type and the provided features, automatically extracts the available image or data sources or generates data for missing required sources. It is also possible to write a complete custom detector by inheriting from the `Detector` class.

In general, it is sufficient to add an LSD class using the template creator functions `createDetectorX`:

```
1     ControlWindow analyzer;
2     // Example for adding line segment detectors to analyzer app
3
```

```
    // Detector provides general source data on its own, use
    // default detector creator
    analyzer.addDetector(
        createDetector<lsfm::LsdBurns<float_type, cv::Point_>>(
            "LSD BURNS", D_MAG_SQR)
    );

    // Detector provides no source data on its own, use ND detector
    // creator, which auto generates the required data (gradients gx,
    // gy, magnitude, dir map, etc.)
    analyzer.addDetector(
        createDetectorND<lsfm::LsdFGioi<float_type, cv::Point_>>(
            "LSD FGIOI")
    );

    // Use a specialized detector generator which can auto extract
    // extended source data
    analyzer.addDetector(createDetectorES<lsfm::LsdEL<
        float_type, cv::Point_, lsfm::Vec2i, lsfm::EdgeSourceQUAD<
            lsfm::QuadratureG2<uchar, float_type>,
            lsfm::NonMaximaSuppression<float_type, float_type, float_type,
                lsfm::FastNMS4<float_type, float_type, float_type>
            >
        >
    >>("LSD EL QFSt Odd"));
```

With this approach, new variants of line segment detectors can easily be added or existing variants can be customized. Similarly, it is possible to add new tools to the analyzer application using the `addTool()` method. An analyzer tool must inherit from the abstract `class LATool` and implement the `connectTools()` method.

```
    // Add a tool to the analyzer by calling .addTool
    analyzer.addTool<ProfileAnalyzer>();
```

More details on how to extend the analyzer application can be found in the readme of the Line Extraction Library GitHub repository [227].

10.4 Evaluation Tooling

The evaluation tool is designed to simplify the setup and execution of experiments, including the comparison of different line extraction methods. It provides a structured pipeline for evaluating the performance or other metrics of different techniques and tools, including semi-automatic report generation of results.

As introduced in the framework design, it builds on top of the given architecture to package individual methods into tasks, which then allow the methods to be included in a test or evaluation case. Multiple cases can be combined and executed with the evaluation pipeline.

The evaluation framework is implemented in the `libs/eval` library and consists of several layers: base task interfaces, input handling, measurement collection, and

result reporting. Each layer adds specific functionality while maintaining loose coupling for flexibility.

10.4.1 Task Architecture

The task architecture (see Figure 10.16) follows a layered design with clear separation of concerns:

- **ITask**: Type-erased interface for Python binding compatibility. Uses `std::any` for input data to provide a stable ABI.
- **Task**: Base implementation providing name, verbose flag, and virtual methods for `prepare()`, `run()`, `reset()`, and `saveVisualResults()`.
- **InputTask<InputDataT>**: Template class linking tasks to typed input data via `DataProvider`. Handles type conversion from `std::any` in `prepareAny()`.
- **MeasureTask<InputDataT, MeasuresT>**: Adds measurement collection capabilities with a `measure()` method and a measures map keyed by source name.

The template parameters allow flexible configuration:

- **InputDataT**: Input data type (must inherit from `GenericInputData`), e.g., `CVPerformanceData` containing a `cv::Mat`.
- **MeasuresT**: Measurement type (must provide a `Result` type), e.g., `CVPerformanceMeasure` storing durations and image dimensions.

A concrete task implementation typically overrides three methods:

```
class MyPerformanceTask : public CVPerformanceTaskBase {
 protected:
  /// Called once per input data to prepare/warm-up
  void prepareImpl(const cv::Mat& src) override {
    // Initialize algorithm, allocate buffers, etc.
  }

  /// Called N times with timing - the actual work
  void runImpl(const std::string& name, const cv::Mat& src) override {
    // Execute the algorithm being measured
  }

  /// Called after all runs to record results
  void measure() override {
    // Store timing data in measures map
  }
};
```

The typical task execution sequence is:

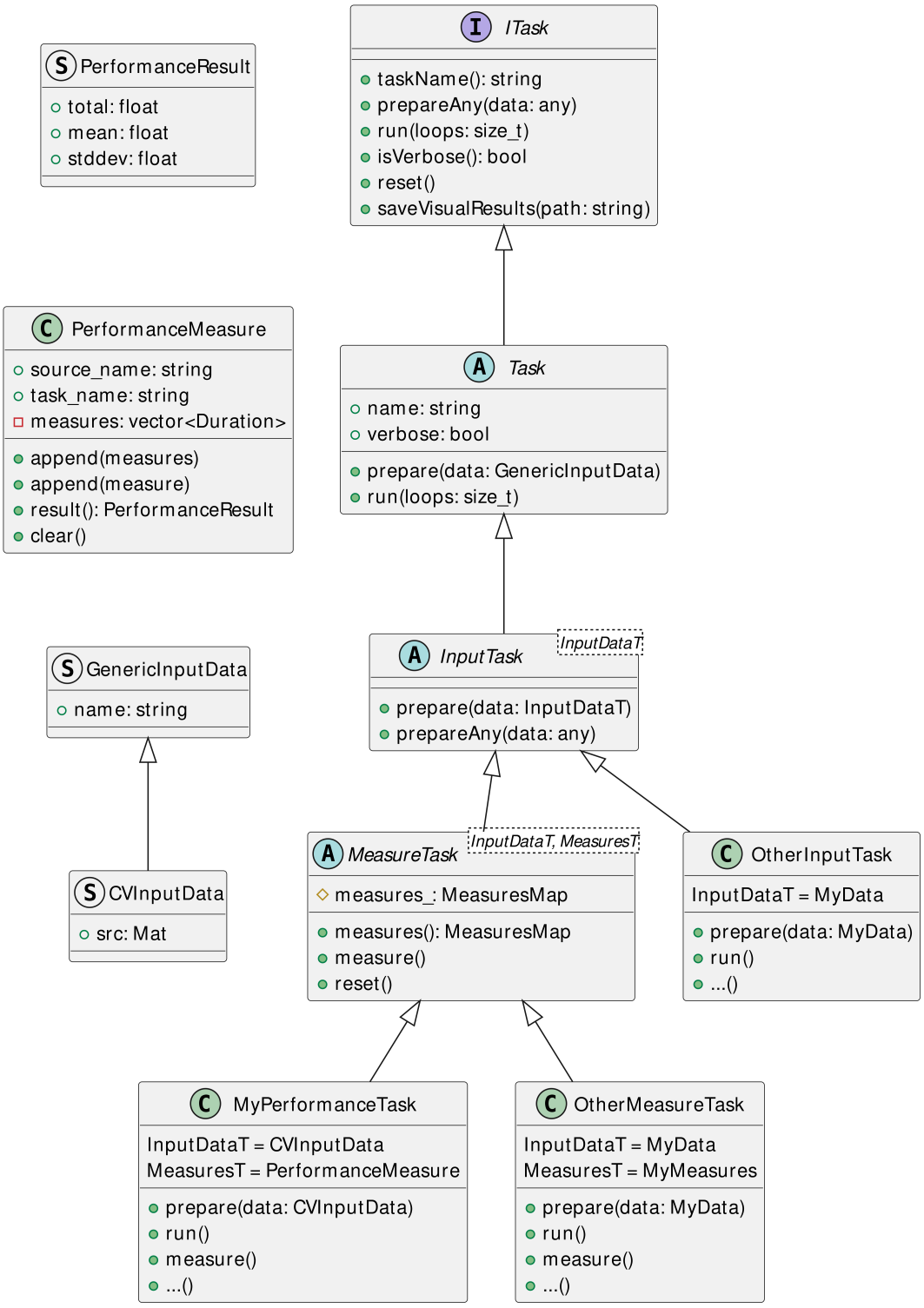


Figure 10.16: Evaluation framework task hierarchy. The ITask interface provides Python binding compatibility, while Task, InputTask, and MeasureTask add layers of functionality.

Sequence 1: Task Execution

Require: Valid input data from DataProvider

1: **prepare(data)**: Load data and prepare task (calls **prepareImpl**)
2: **for** $i = 1$ to *loops* **do**
3: **run()**: Execute task with timing (calls **runImpl**)
4: **end for**
5: **measure()**: Record timing results to measures map

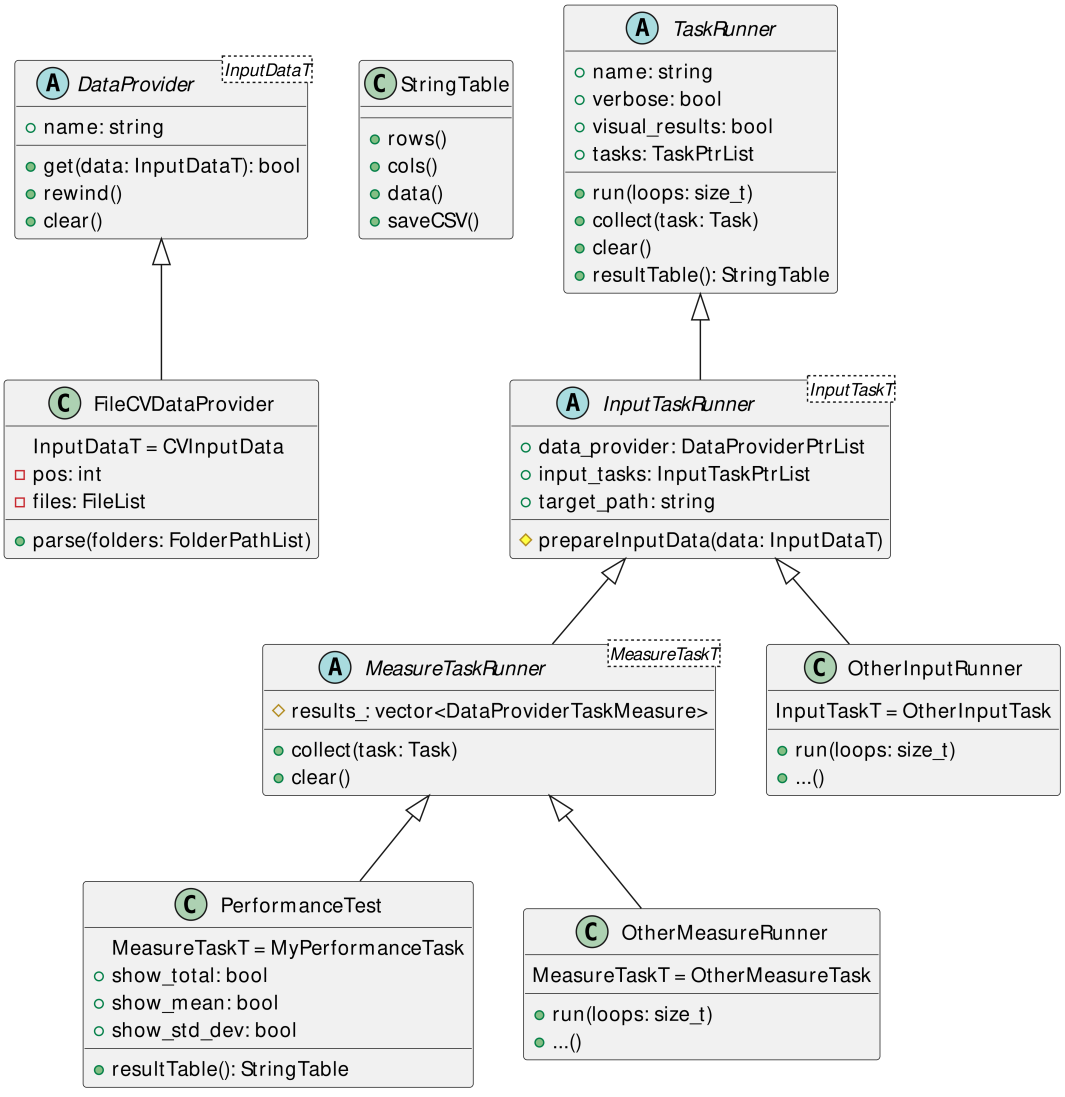


FIGURE 10.17: Task runner hierarchy. `TaskRunner` provides the base, `InputTaskRunner` adds data providers, and `MeasureTaskRunner` handles result collection. `PerformanceTest` specializes for timing measurements.

10.4.2 Task Runner Architecture

Once tasks are implemented, they are combined into test cases using task runners (see Figure 10.17). The runner hierarchy mirrors the task hierarchy:

- **TaskRunner**: Base class with task list, `run()`, `collect()`, `clear()`, and `resultTable()` methods.
- **InputTaskRunner<InputTaskT>**: Adds a list of `DataProvider` instances for feeding input data to tasks.
- **MeasureTaskRunner<MeasureTaskT>**: Collects measurements from tasks after execution and stores them for result generation.
- **PerformanceTest<PerformanceTaskT>**: Specialized runner for timing measurements with configurable output (total, mean, standard deviation, megapixels).

The `MeasureTaskRunner` execution flow processes all data providers and tasks:

Sequence 2: Task Runner Execution

```
Reset all tasks
for each data_provider in data_providers do
    data_provider.rewind()
    for each data in data_provider do
        for each task in tasks do
            task.prepare(data) {Prepare with new input}
            task.run(loops) {Execute N iterations}
            task.measure() {Record results}
            collect(task) {Aggregate measures}
        end for
    end for
end for
```

A complete performance test setup looks like:

```cpp
// Create data provider for test images
auto provider = std::make_shared<FileDataProvider>(
    "/path/to/images", "TestImages");

// Create performance test with data provider
auto test = std::make_shared<CVPerformanceTest>(
    DataProviderList{provider}, "NMS Comparison");

// Add tasks to compare different NMS implementations
test->input_tasks.push_back(std::make_shared<NMSTask<FastNMS4>>("FastNMS4"));
test->input_tasks.push_back(std::make_shared<NMSTask<FastNMS8>>("FastNMS8"));
test->input_tasks.push_back(std::make_shared<NMSTask<PreciseNMS>>("PreciseNMS"));

// Configure output options
test->show_mean = true;
test->show_std_dev = true;

```

```
// Run with 10 iterations per image
test->run(10);

// Generate and export results
StringTable results = test->resultTable();
results.saveCSV("nms_comparison.csv");
```

The `resultTable()` method aggregates all measurements into a `StringTable` that can be exported as CSV for further analysis or inclusion in reports.

10.4.3 Eval Application

The performance evaluation application in `evaluation/performance/` demonstrates the framework in action. It measures the runtime of various algorithms implemented in this thesis, from gradient computation to line segment detection.

Each test file (e.g., `nms.cpp`, `gradient.cpp`, `lsd.cpp`) defines task classes that inherit from `CVPerformanceTaskBase`. This base class provides:

- **prepareImpl(src)**: Called once per image for warmup and initialization
- **runImpl(name, src)**: Called N times with automatic timing
- **measure()**: Records timing to the measures map

A test creator function registers each test with a factory:

```
// In nms.cpp - define NMS performance tasks
class NMS4Task : public CVPerformanceTaskBase {
 protected:
  void prepareImpl(const cv::Mat& src) override {
    // Compute gradients for NMS input (not timed)
    gradient_.process(src);
  }

  void runImpl(const std::string&, const cv::Mat&) override {
    // Only NMS execution is timed
    nms_.process(gradient_.gx(), gradient_.gy(), gradient_.magnitude());
  }
};

// Register test with factory
CVPerformanceTestPtr createNmsTest(DataProviderList providers) {
  auto test = std::make_shared<CVPerformanceTest>(providers, "NMS");
  test->input_tasks.push_back(std::make_shared<NMS4Task>("FastNMS4"));
  test->input_tasks.push_back(std::make_shared<NMS8Task>("FastNMS8"));
  test->input_tasks.push_back(std::make_shared<NMSPreciseTask>("PreciseNMS"));
  return test;
}
REGISTER_PERFORMANCE_TEST("nms", createNmsTest);
```

The main application uses the factory to run selected tests:

```cpp
int main(int argc, char* argv[]) {
  // Parse command line options
  auto options = parseOptions(argc, argv);

  // Create data providers from image directories
  DataProviderList providers;
  for (const auto& dir : options.image_dirs) {
    providers.push_back(std::make_shared<FileDataProvider>(dir, dir));
  }

  // Run selected tests
  for (const auto& test_name : options.tests) {
    auto test = PerformanceTestFactory::create(test_name, providers);
    test->run(options.loops);

    // Output results
    auto table = test->resultTable();
    std::cout << table.format() << std::endl;
    table.saveCSV(test_name + "_results.csv");
  }
}
```

The application supports command-line configuration for test selection, number of iterations, output formats, and data directories. Test configurations can also be saved to and loaded from files.

10.5 Python Tools

Most of the results presented in this work were initially implemented using the approach described in the previous section, supported by Python tools for generating tables and graphs (similar to those included in this thesis, although native LaTeX functions were used for the output generation). While these tools proved to be valuable during development, they offered limited flexibility and integration capabilities. To address this, all C++ library modules have been fully exposed to Python via Pybind11[1] bindings. This allows tasks, test cases, and entire test suites to be created directly in Python, while the underlying algorithms are still executed at native C++ speed.

Making the algorithms and methods available in Python provides two main benefits:

1. Algorithms can be used directly and efficiently in experiments without leaving the Python ecosystem.
2. New algorithmic variants can be added by implementing derivatives on the Python side, while still being executable at native C++ side by embedding a Python runtime within a C++ host if required.

10.5.1 Python Binding Modules

Every library in the framework has a corresponding Python module, compiled as a native extension and importable via a consistent `le_*` naming convention

[1] https://pybind11.readthedocs.io/en/stable/

(le_imgproc, le_edge, le_geometry, le_lsd, le_lfd, le_eval, le_algorithm). The bindings are built alongside the C++ targets using Bazel and placed into bazel-bin/libs/{module}/python/. The full C++ class hierarchy of each library — including all interfaces, concrete implementations, and template specializations — is exposed, so all methods discussed in the preceding sections are available from Python with identical semantics.

A few notable additions or adaptations exist on the Python side:

- **le_lsd** adds the free function fit_line_segments(), which converts OpenCV-style contour arrays (as returned by cv2.findContours) directly to LineSegment lists — enabling tight integration with Python-based segmentation workflows.
- **le_eval** exposes the full benchmarking pipeline described in section 10.4, including Python-subclassable task base classes with overridable prepare_impl and run_impl hooks so that new benchmark tasks can be written entirely in Python.
- **le_algorithm** additionally provides tools intended for interactive Python use: hyperparameter search (ParamOptimizer with grid and random strategies), detector profiling (ImageAnalyzer, DetectorProfile), and configuration management (PresetStore).

In addition to the native binding modules, a pure-Python utility package lsfm (located in python/lsfm/) provides higher-level helpers: a complete stereo reconstruction pipeline (reconstruction.py), synthetic image generation (synthetic/), and notebook utilities (notebook.py).

10.5.2 Binding Technique

All bindings are implemented in a per-library python/src/ directory using Pybind11. Because many C++ classes in the framework are templated, the binding layer uses a two-step approach: a binding function is parameterized by the relevant template arguments, and the module initialization instantiates it for each supported type combination. A type-suffix convention (e.g. "" for the default uchar/float preset, "_f32" for explicit 32-bit floating-point) disambiguates the resulting Python class names:

```
// Template binding function for one type preset
template <class IT, class GT, class MT, class DT>
void bind_edge_preset(py::module_& m, const std::string& suffix);

PYBIND11_MODULE(le_edge, m) {
    bind_edge_core_types(m);
    // Default preset: uchar image, float gradient/magnitude/direction
    bind_edge_preset<uchar, float, float, float>(m, "");
    // Explicit 32-bit floating-point preset
    bind_edge_preset<float, float, float, float>(m, "_f32");
}
```

Standard C++ containers (std::vector, std::map) are handled transparently by Pybind11's built-in STL adapters. For cv::Mat, the project uses *cvnp*[2] — a dedicated

[2] https://github.com/pthom/cvnp

Pybind11 type-caster library that maps `cv::Mat` to and from NumPy arrays via shared memory, so no data is copied at the Python boundary. Because the `le_*` modules and the official OpenCV Python package (`cv2`) both exchange data as standard NumPy arrays, they are fully interoperable: images loaded with `cv2.imread()` can be passed directly to any `le_*` function. Known pitfalls and workarounds are documented in the framework readmes.

Usage Example

The complete edge detection and line segment extraction pipeline can be driven entirely from Python in just a few lines:

```python
import cv2
import le_imgproc, le_edge, le_lsd, le_geometry

# Load image
img = cv2.imread("image.jpg", cv2.IMREAD_GRAYSCALE)

# Run LsdEL detector (edge-linking based, most feature-rich)
det = le_lsd.LsdEL()
det.detect(img)
segments = det.line_segments()

# Draw results with random colors
canvas = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)
le_geometry.draw_lines_random_inplace(canvas, segments)

# Access intermediate edge data directly from the detector
magnitude = det.magnitude()    # gradient magnitude map (NumPy array)
direction = det.direction()    # edge direction map (NumPy array)

print(f"Detected {len(segments)} line segments")
```

Custom performance tasks that extend the C++ benchmarking framework can be implemented in Python by subclassing `CVPerformanceTaskBase`:

```python
import le_eval, le_imgproc
import numpy as np

class GradientTask(le_eval.CVPerformanceTask):
    """Benchmark task for gradient computation."""

    def __init__(self, name: str, gradient_cls) -> None:
        super().__init__(name, le_eval.TASK_SQR)
        self.gradient_cls = gradient_cls
        self.grad = None

    def prepare_impl(self, src: np.ndarray) -> None:
        self.grad = self.gradient_cls()

    def run_impl(self, name: str, src: np.ndarray) -> None:
        self.grad.process(src)
```

```
17
18 # Run benchmark and export CSV
19 provider = le_eval.FileCVDataProvider("/path/to/images", "test")
20 test = le_eval.CVPerformanceTest([provider], "Gradient Comparison")
21 test.add_task(GradientTask("Sobel", le_imgproc.SobelGradient))
22 test.add_task(GradientTask("Scharr", le_imgproc.ScharrGradient))
23 test.run(loops=10)
24 test.result_table().save_csv("gradient_benchmark.csv")
```

10.5.3 Jupyter Notebooks

All binding modules are documented and demonstrated through a collection of Jupyter Notebooks[3] located in `examples/notebooks/`. The notebooks are integrated into the project's Python environment (managed via `uv` and `pyproject.toml`) and can be opened directly in VS Code or JupyterLab. Two introductory notebooks serve as entry points:

- A self-contained computer vision primer (no library bindings required) covering images as arrays, gradient computation, edge detection, and the LSD taxonomy.
- A compact API tour of all binding modules with inline visualizations and performance comparisons.

A structured tutorial series then covers each part of the framework in depth:

1. Library fundamentals: gradient filters, geometry primitives, and configuration (`le_imgproc`, `le_geometry`).
2. Edge and line detection pipelines: edge sources, NMS, and all nine LSD detectors with noise robustness analysis (`le_edge`, `le_lsd`).
3. Performance evaluation framework: data providers, custom Python tasks, and result export (`le_eval`).
4. Algorithm library: post-processing, accuracy evaluation, hyperparameter search, and sub-pixel refinement (`le_algorithm`).
5. Advanced filters: SUSAN, RCMG color gradient, and quadrature filters with parameter sweeps (`le_imgproc`).
6. Image operators and pipelines: operator subclasses, `PipelineOperator` composition, and augmentation workflows (`le_imgproc`).
7. 3D geometry and camera models: projection, stereo triangulation, and Rerun.io 3D visualization (`le_geometry`).

Beyond the tutorials, several demo notebooks illustrate practical use cases such as Rerun.io-based visualization, line feature matching, PyTorch/SAM segmentation integration, and stereo 3D line reconstruction.

This tooling directly integrates the line extraction library into Python-enabled AI and visualization frameworks, enabling exploratory and interactive workflows that are not possible with the pure C++ tooling alone.

[3] `https://jupyter.org/`

10.6 Further Information

For more information such as installation instructions, usage examples, API documentation or licensing, please see the readme of the Line Extraction Library GitHub repository [227].

Chapter 11

Outlook

Of course, this work still leaves many open questions. For example, further investigation to find the best methods to operate on highly noisy images, or optimization methods to find optimal thresholds or NFA values to reduce unimportant information. Moreover, further and more extensive experiments can be conducted to derive best practices or to gain further insight into which method is best suited for which application area. The framework presented in this thesis provides excellent tools to get started right away to answer those questions. With the ability to integrate methods and exchange data via Python, it also invites the integration of other approaches, such as modern AI-based methods, and their inclusion in the evaluations.

Further work on line features, particularly in the areas of descriptors, matching, and tracking, was done by Manuel Lange. I have also contributed to these topics in various ways, but my main focus has been on efficient line segment extraction, as demonstrated in this work, and by providing the extensive open source library as a tool [227] to facilitate future research using edge and line segment information.

Efficient line segment extraction is important to the overall process of using line features, as accurate and efficient extraction directly affects the quality of the subsequent feature description and matching results. For a more detailed analysis and results regarding line features, I refer the reader to Manuel Lange's thesis [118].

The efficient line segment extraction was also intended to be used for 3D reconstruction of large-scale areal environments from lines. Reconstruction from lines has gained attention as an alternative or complementary approach to traditional point-based methods, particularly in urban and man-made environments where structures often consist of straight edges and planar surfaces. Line-based reconstruction provides richer geometric constraints than point-based methods, especially in scenarios with repetitive textures, low-texture regions, or occlusions.

Recent advancements in both areas have been achieved through AI-driven approaches. Learning-based line descriptors and 3D reconstruction methods now outperform traditional techniques.

11.1 Line Features

In his thesis "Visual Odometry Using Line Features and Machine Learning Enhanced Line Description", Manuel Lange also presented an AI-based line descriptor that significantly improved the accuracy of line feature matching. Using machine learning techniques, specifically a modified ResNet architecture, that was trained with

synthetic data generated with the Unreal Engine 4[1], this approach outperformed traditional methods such as the Line Band Descriptor (LBD) [236].

In addition to Manuel Lange's work, several AI-based approaches have been developed to improve line feature descriptors and matching. For example, Yoon et al. [232] proposed a context-aware line descriptor using transformer architectures that consider line segments as sequences of points. This method dynamically considers significant points along a line and can effectively handle variable line lengths and improve matching accuracy.

Another notable approach is that of Kannapiran et al. [102], who introduced a stereo visual odometry technique that uses an attention-based graph neural network for robust point and line feature matching. This method is particularly effective under varying weather and dynamic lighting conditions.

11.2 3D Reconstruction

Recent AI-driven approaches can also detect, extract, and reconstruct 3D structure from line-based features. Unlike traditional edge-based techniques that rely only on feature extraction and vanishing point estimation, deep learning models can infer depth and structure from lines even in ambiguous scenarios. For example, learning-based methods have been applied to monocular 3D line reconstruction, where convolutional neural networks (CNNs) predict depth information from detected line segments. This approach greatly improved the accuracy of reconstruction in structured environments [94].

Neural network-based structural constraints or priors can be used to improve line-based reconstruction by encoding the regularities of architectural structures and objects into deep models. These methods help to resolve occlusions or missing lines by using the learned shape priors. This makes them useful in urban scene reconstruction and CAD model generation [238]. In addition, hybrid approaches that integrate points and lines into a joint optimization framework can provide more accurate reconstructions. This is in particular true for indoor and outdoor architectural scenes, where lines define walls, windows, and other structural elements more effectively than points alone [91].

A graph-based deep learning approach has also been applied to line-based 3D reconstruction. In this case, a graph represents structural relationships between edges, allowing AI models to infer missing or occluded structures by using connectivity constraints [99]. These approaches can also greatly improve the completeness and realism of 3D reconstructions, again notably in structured environments where geometric consistency is important.

11.3 AI Approaches

Also advancements using AI on resource-constrained platforms, such as drones, have improved. Optimized AI models and better hardware accelerators allow real-time

[1] https://www.unrealengine.com/

processing in environments with limited computing resources, memory, and power. For example, GAP9Shield, a 150 GOPS-capable AI processor, enables on-board vision and navigation capabilities for ultra-low-power systems [156]. Together with Open-source frameworks such as ADAPT, which integrates AI processing modules, enables real-time decision making on compact embedded systems [206].

Optimized AI models, such as pruned and quantized deep networks, reduce computational complexity while maintaining accuracy [166]. This improves efficiency and enables AI inference on low-power platforms. Adaptive deep learning further adjusts model complexity based on available resources [154]. This ensures that the performance adapts to changing computational constraints.
With this development, AI-based line descriptors and matching techniques become more attractive for resource-constrained devices because they now offer real-time capability. Traditional methods may become less relevant as AI-driven approaches achieve better performance.

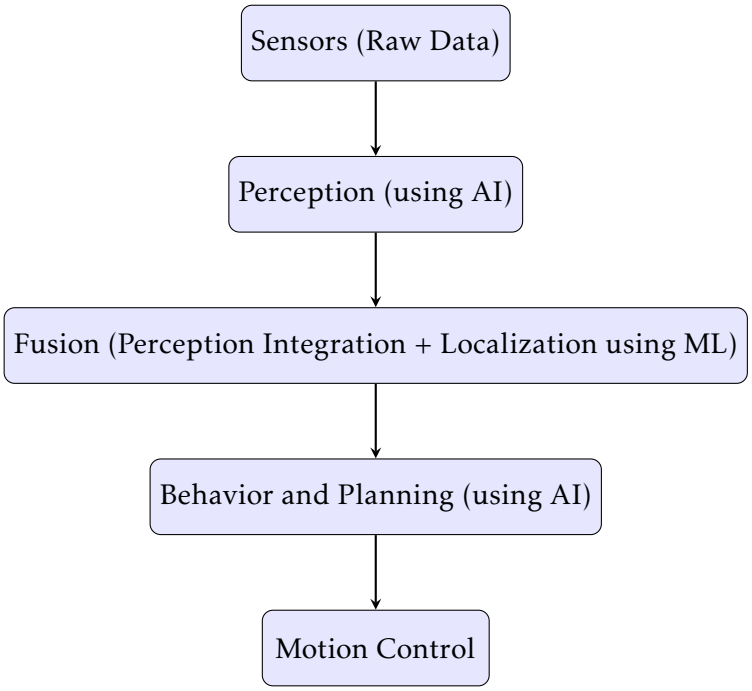


Figure 11.1: Traditional autonomous vehicle software pipeline with hierarchical layout.

A notable trend in this field, observed both in research [124, 11] and in industry (including first-hand insights from my company), is the increasing adoption of end-to-end AI approaches. In areas such as autonomous vehicles, software architectures are shifting from AI applied to specific subproblems to AI that handles the entire pipeline (see Figure 11.1 and Figure 11.2).

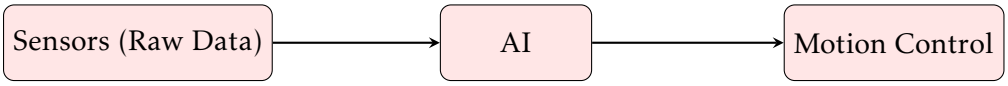


Figure 11.2: Latest AI-based autonomous vehicle software pipeline.

Improved traceability and validation capabilities of AI help to address safety and regulatory concerns. AI that is easier to interpret also increases confidence in its use. This is particularly important for applications that need to go through an elaborate certification process, such as autonomous driving functions [104, 165]. These advances make it more likely that the first official approvals for AI-controlled systems on the road will soon be granted.

While new projects are likely to adopt AI-based approaches from the beginning to address complex vision problems, explicit line and edge extraction still offers distinct advantages. Line segments are highly interpretable, geometry-based features that enhance the explainability and traceability of AI pipelines. In safety-critical domains such as autonomous driving, geometric constraints derived from line extraction can validate or cross-check AI predictions.

Combining line extraction with AI models can help to focus the processing on geometrically relevant regions, reducing the search space and computational cost. For example, AI can classify which extracted line segments are relevant for precise measurements, structural inspections, or mapping tasks.

Hybrid pipelines can leverage the strengths of both paradigms: AI provides flexible learning capabilities, while classical line extraction ensures a stable geometric foundation. In resource-constrained systems, lightweight line-based algorithms combined with adaptive AI modules can achieve robust performance while keeping computational demands low.

In certain cases, such as processing extremely high-resolution images where accuracy is critical, traditional methods may still outperform AI solutions, which may struggle with large data volumes or fail to meet required precision levels.

Finally, the educational value of classical methods should not be underestimated. The GUI application presented in the previous chapter (see section 10.3) encourages experimentation with different edge- and line-processing approaches. Moreover, a solid understanding of these techniques remains essential for correctly interpreting and evaluating AI-based results in this domain.

Appendix A

Appendix

A.1 DC Component

A common problem with creating symmetric kernels for convolutions is the DC component [199, 147, 31]. The DC should be zero, meaning that the average or sum of the filter signal should be zero:

$$\sum_{i=0}^{n} k(i) = 0 \tag{A.1}$$

If it is not close to zero, artifacts will appear in the filtered images, as shown in Figure A.1 (left column). The main causes of DC problems are aliasing effects or early clipping of the filter signal. To mitigate this problem, the kernels must be large enough. Therefore, in most cases, it makes sense to perform the computation in the frequency domain, since Fourier transform and multiplication are more efficient than convolution with large kernels. In addition, the filter can be applied without truncating the filter signal. However, aliasing effects may still occur.

For small kernels, convolution can be much faster. To still get good results, the DC must be fixed. This can be done by calculating the mean of the kernel k and subtracting the mean from k:

$$\begin{aligned} k_{mean} &= \frac{\sum_{i=0}^{n} k(i)}{n} \\ k_{zeroDC} &= k - k_{mean} \end{aligned} \tag{A.2}$$

The variable n is the number of elements in the kernel, e.g. for a 3x3 kernel $n = 9$. In some cases, such as the quadrature filter pairs (see subsection 4.2.1), the DC correction can have some side effects, as discussed in the next section.

Another solution, which is not always possible, is to find the optimal spacing parameters of the filter signal for a given filter size. To tune the parameters, an optimization method such as the Levenberg-Marquardt algorithm [123] can be used. The algorithm requires a continuous error function given by the absolute value of the kernel mean:

$$k_{err} = \left| \frac{\sum_{i=0}^{n} e(i)}{n} \right| \tag{A.3}$$

If multiple filters are involved, as in the case of the steerable quadrature filter (see subsection 4.2.3), another error function can be used. When the complex filter is applied to a homogeneous image, the resulting response should be zero. Therefore,

the absolute value of the response can also be used as the error function.

Since this thesis requires corrections for kernel operators, particularly in the edge region detector comparison chapter (see chapter 5), either the mean-kernel fix is applied to correct the DC component or an optimization method is used to determine the optimal kernel spacing for the filters. To illustrate the effects, a small study was initially conducted in the comparison chapter and later moved to the appendix.

Method	3x3	5x5	7x7	9x9
LoG	1.240	1.008	0.873	0.782
QF St	1.240	1.008	0.873	0.782
SQF PO	2.621	2.209	1.987	1.844

Table A.1: Kernel spacing parameters estimated with the optimization method.

Method	3x3	5x5	7x7	9x9
LoG	0.155	0.014	0.001	0.000
QF StG	0.142	0.013	0.001	0.000
SQF PO	0.107	0.096	0.087	0.080

Table A.2: Remaining error (average of kernel) with optimal kernel spacing.

Table A.1 shows the optimal kernel spacing parameters for different kernel sizes. The remaining error (average of the kernel or response of homogeneous images) with optimal kernel spacing is shown in Table A.2. For the spherical quadrature filter with Poisson kernel (SQF PO, see subsection 4.2.5), the scale and scale multiplier parameters have been set to $scale = 1$ and $muls = 2$. Adding them to the optimization process will fail because the resulting error function won't be continuous anymore.

The first table shows that the optimized kernel spacing parameters for the Laplacian of Gaussian (LoG, see subsection 4.1.3) and quadrature based gradients (QF StG, see subsection 4.2.3, for an overview of filters refer to chapter 5) are the same. Since the QF StG uses three directed Gaussian second derivative kernels to compute the responses and then steers to the direction with the maximum local energy, the resulting response of the even part is quite similar to the LoG response, since both methods use Gaussian derivatives of the simplified Gaussian equation as defined in subsection 2.2.1.

What can be seen from the error values in Table A.2 and Figure A.3 is the requirement for sufficiently large kernels. For small kernels, the error is high, even with optimized kernel spacing parameters. This is due to the fact that the sampling rate is too low, causing strong aliasing effects and premature truncation of the signal. Applying the DC correction helps to reduce the artifacts, but also modifies the initial filter function quite strongly. This can lead to other side effects of operations that depend on the resulting data of the filter.

Figure A.1 shows an example set of even responses of the steerable quadrature filter of a simple disk (see also top left of Figure A.2). It shows that even for larger filter

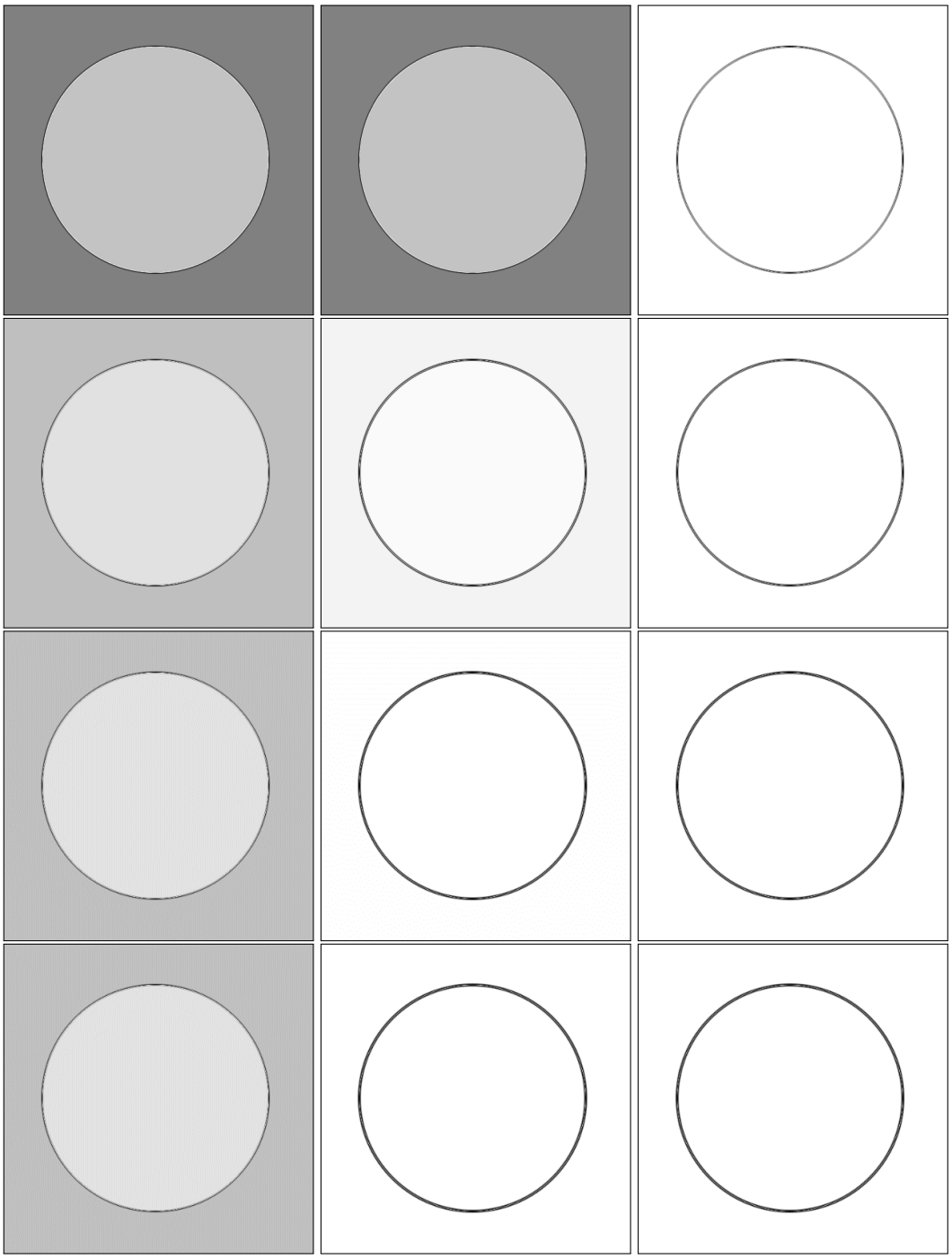


FIGURE A.1: This figure shows the even responses of the steerable quadrature filter of a simple disk image. Each row represents a different kernel size of the operator (3x3, 5x5, 7x7, and 9x9). The first column shows the filter responses for a fixed kernel spacing of 1.2. The second column shows the filter responses with the optimized kernel spacing, and the last column shows the optimized filter responses with additional zero DC correction.

kernels, the non-zero DC leads to non-optimal responses (gray instead of black on the left and right of the disk edge responses). The optimized kernel spacing parameters reduce the error, especially for larger kernel sizes. However, the DC correction still helps to further improve the response and is mandatory for small kernel sizes.

A.2 Phase

The phase is calculated from the even and odd parts of a quadrature filter pair (see subsection 4.2.1). Applying a DC correction, as introduced in the previous section, will affect the odd filter response and therefore the phase.

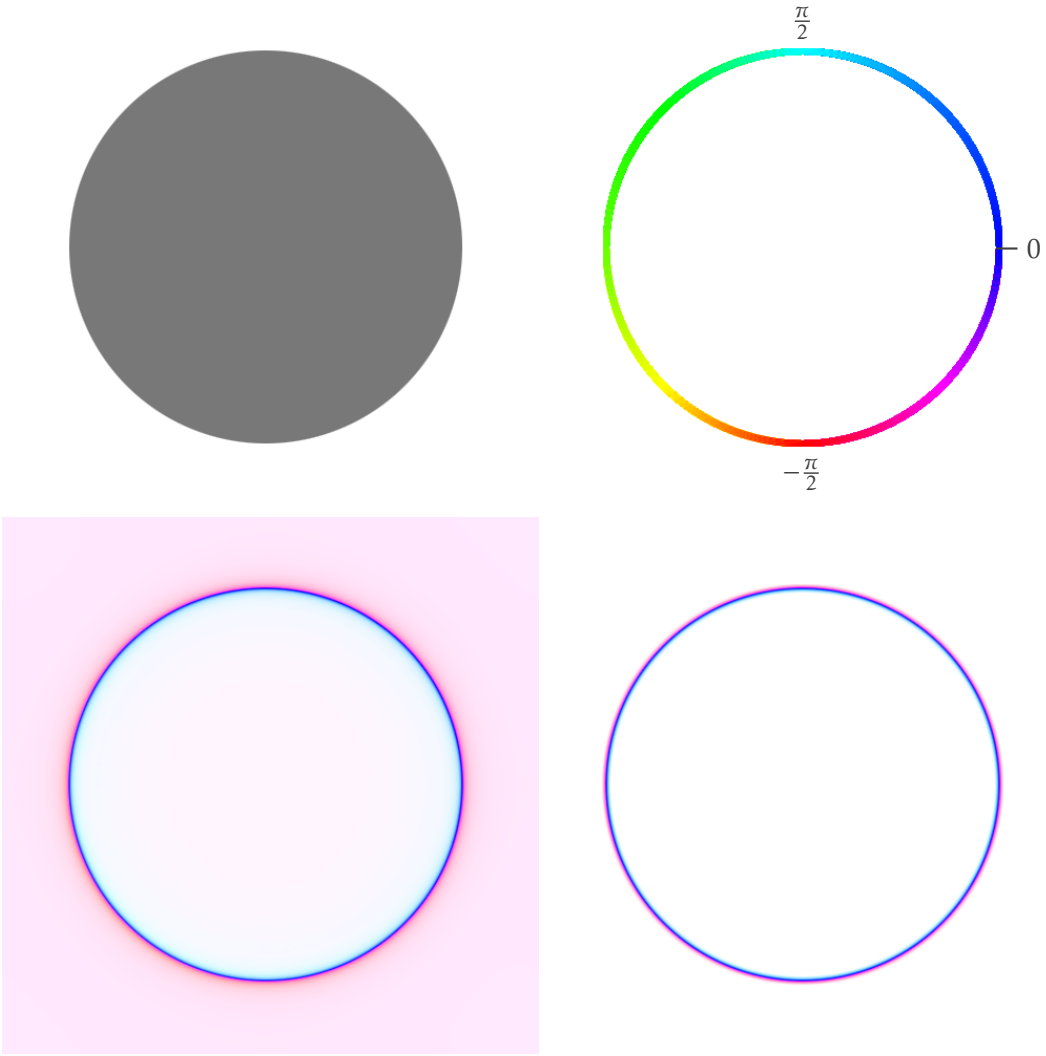


FIGURE A.2: Phase example: Top left: Example disk image; Top right: Angle color coding used for phase; Bottom left: Phase weighted by energy response of the disk image via SQFF; Bottom right: Phase via SQF 9x9 also weighted by energy response.

Figure A.2 shows the phase responses of the dark disk. The top row shows the input image and the angular color-coding for the phase, while the second row visualizes the color-coded phase responses weighted by the local energy response. The weighting

is used to hide or reduce the visibility of the non-significant phase response. Both responses show quite similar phase results at the edge location. The visualized phase response of the left image was computed with the filter in the frequency domain and radiates far from the edges, even at the location where the weighted phase response should be zero. While the filter signal is not truncated in this case, resulting in a more accurate and spread out phase response at the edge locations, aliasing effects can still occur, explaining the erroneous behavior. The phase response in the right image is computed with a truncated filter in the spatial domain. The applied DC correction prevents strong artifacts, and the truncation of the filter signal reduces the strong spread at the edge locations, but also affects the phase response at the edges.

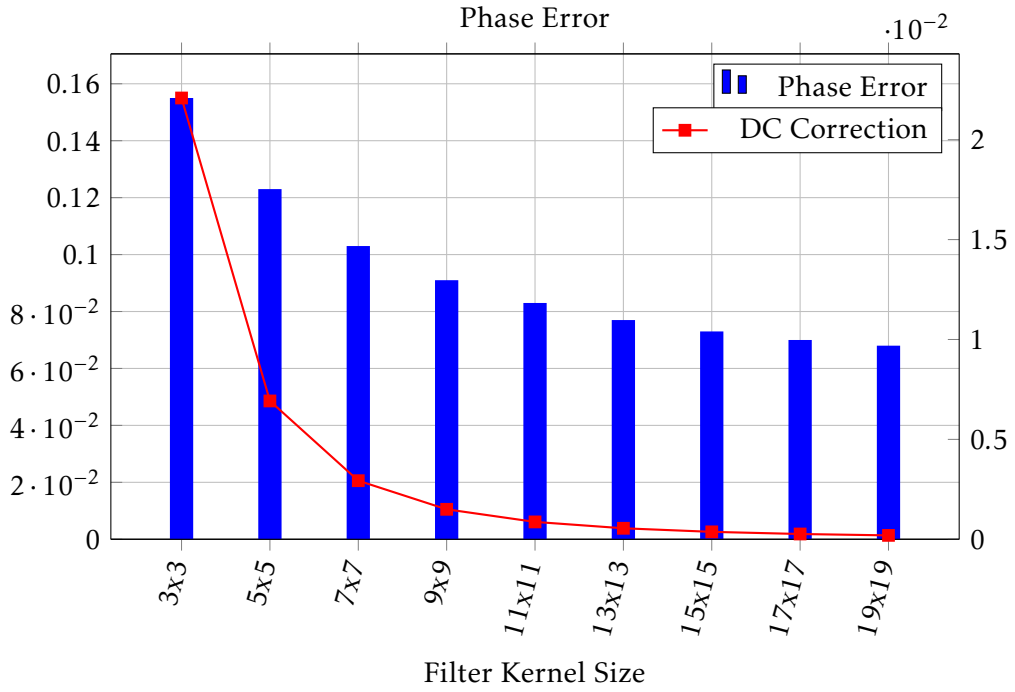


Figure A.3: Illustrates the results of the phase differences (bars) and the DC correction values (line) that were applied to the convolutional filter.

To measure the influence, the difference between the phase of the truncated spatial domain SQF filter with DC correction and the phase of the frequency domain SQFF filter is computed. Additionally, a local energy dependent mask is applied to include only the phase information at the edges.

The diagram in Figure A.3 illustrates the results of the phase differences and also includes the DC correction applied to the convolutional filter. It shows a decreasing phase error for larger filter kernels. The larger the kernel, the less significant is the DC correction value required to fix the filter kernel. Thus, as the filter size increases, the phase of the truncated filter in the spatial domain approaches the phase of the untruncated filter. The phase error in the edge region remains quite small even for small filter mask sizes.

List of Figures

1.1 Cell types in different layers of a human retina. R: rods; C: cones; H: horizontal cells; Bi: bipolar cells; A: amacrine cells; G: ganglion cells. 3
1.2 At the edges of grayscale bars with different intensities from dark to light, the human eye interprets a non-existent gradient. By applying the Laplacian of Gaussian operator to the Mach band image, the edges are enhanced, making the image sharper and separating each gray bar from the others by overshooting and undershooting at each step of intensity change. 4
1.3 On the right side, the 1D intensity function is plotted for the selected scan line of the image on the left side. 6
1.4 Examples of step and ramp edges as ideal and noisy signals. 6
1.5 Examples of ideal and noisy peak, ridge, and roof edge signals. . . . 7

2.1 : Example for a mean/box filter on the left side and a Gaussian filter on the right side. 11
2.2 Examples of smoothing filters from left to right: Original image; Box filtered image; Gaussian filtered image. The mask size for both filters was 9x9. 11
2.3 Derivative filter examples from left to right: Original image, Sobel filter and Laplacian filter. The mask size for both filters was 3x3. For the Laplacian image, the absolute filter response was used. 14
2.4 First row from left to right: original image, Gaussian smoothed image, stronger Gaussian smoothed image. Second row from left to right: absolute difference of the first two images in the first row, absolute difference of the second two images in the first row. 16

3.1 Top left: Example color image; Top right: Canny of grayscale image; Bottom left: Canny of color image; Bottom right: Difference of canny edge images 22
3.2 This chart illustrates the minimum error values for each method of the given parameter set. The errors are expressed relative to the real error (first row in Table 3.3): 1 no improvement, 0 perfect denoising. . . . 25
3.3 Illustrates the execution time (ms) for the denoising filtering methods with increasing filter size (see section 5.1 for hardware setup). 26
3.4 Example images to visualize the denoising results. A Gaussian noise with an intensity of 20 was applied before denoising. 27
3.5 Colorized edge maps (by orientation) to visualize the edge results after denoising. 28

4.1 Plot of Gaussian function and its first derivative with $\sigma = 1$. 32
4.2 Plot of Deriche function and its first derivative with $\alpha = 1$. 33
4.3 Plot of Shen et al. function and its first derivative with $\alpha = 1$. 33
4.4 Cosine (even) and sine (odd) wave modulated by a Gaussian. 36

4.5 On the left is an image with a transition between line and step edge. On the right, the 1D intensity functions corresponding to the image are plotted. 37
4.6 Edge response of the Gaussian first derivative operator on the left; local energy response of the steerable Gaussian quadrature operator on the right. Both using a 5x5 filter kernel. 37
4.7 NMS of Sobel operator (left) and local energy (right). 37
4.8 Illustration of the phase congruency components: The final responses $F(x)$ and $H(x)$ are the sum of the even and odd filter responses $e_n(x)$ and $o_n(x)$ of different scales, while the amplitude $a_n(x)$ and phase $\varphi_n(x)$ are the polar coordinates of the even and odd filter responses of scale n. 43

5.1 Chart of fast methods that can run in real time even for high-resolution images on the test system. 56
5.2 Chart of methods that can run in real time for low and medium resolution images on the test system. 56
5.3 Chart of slow methods, some of which can run in real time on low-resolution images on the test system. The PC ML, RMG 5x5, and RCMG methods are not suitable for real-time use even at low resolution without further optimization or faster hardware. 57
5.4 Chart of execution time (ms) for filters with increasing kernel size for low-resolution images. 57
5.5 Chart of filter runtime (ms) with increasing kernel size for high-resolution images. 58
5.6 Chart of Gaussian blur runtime (ms) with increasing kernel size for half resolution images. 60
5.7 Chart of Gaussian Blur runtime (ms) with increasing image size. . . 61
5.8 Chart of FFT runtime (ms) with increasing image size. 62
5.9 Example image to visualize the edge response results. 63
5.10 Edge responses of bicycle2 image part 1 - Left: without noise, Right: with noise. 64
5.11 Edge responses of bicycle2 image part 2 - Left: without noise, Right: with noise. 65
5.12 Edge responses of bicycle2 image part 3 - Left: without noise, Right: with noise. 66
5.13 Edge responses of bicycle2 image part 4 - Left: without noise, Right: with noise. 67
5.14 Edge responses of bicycle2 image part 5 - Left: without noise, Right: with noise. 69
5.15 Illustration of edge responses using different kernel sizes part 1 - Left column: QF StG method, Right column: LoG method. 70
5.16 Illustration of edge responses using different kernel sizes part 2. . . 71
5.17 Average absolute differences between original image and noisy images (noise 10, 20, 30, 40 and 50) 72
5.18 Noise sensitivity behavior as filter kernel size increases. 72
5.19 Top left: Box images with rotations of 0°, 15°, 30°, and 45°. Upper right: Selection mask for box images. Bottom left: Disk image. Bottom right: Disk image selection mask. The red arrows indicate the direction of the ground truth gradient. 73

5.20 Left: The four test variants for the box and disk (normal, smooth edges, noise, blur and noise). Right: Example visualization of orientation error using the Scharr operator. 74
5.21 Bar chart for the orientation errors of box setup containing the four test variants (normal, blurred, noise 10 and both) and the average error. 77
5.22 Bar chart for the orientation errors of disk setup containing the four test variants (normal, blurred, noise 10 and both) and the average error. 78
5.23 Shows the mean radian error of both setups using all variants depending on the filter kernel size. 78
5.24 Mean error for no noise, images with low noise (mean 10) and high noise (mean 40). The mean error is based on both test setups (box and disk), both variants (no blur, with blur). 79

6.1 Threshold example using the Otsu threshold estimator (see subsection 6.1.4). 82
6.2 Threshold example using low, good and high threshold values. . . . 83
6.3 Threshold examples using a global threshold method (left), a local threshold method (center, 3x3 tiles) and dynamic threshold (right, radius for threshold estimation: 50 pixels). All methods used the Otsu threshold estimator. 84
6.4 Example of an erosion operator applied (OpenCV implementation based on [237]) (right) to a binary edge response image (left). 86
6.5 This figure shows a magnitude snippet of the windmill with two cases for NMS. Large green circle: Real maximum. The arrow shows the gradient direction (which is perpendicular to the edge orientation) and strength, the smaller darker circles show the selected neighbors. Large red circles: Suppressed edge response because there is no maximum between the neighbors selected based on the gradient. 87
6.6 NMS examples using double threshold (hysteresis) to extract the final edge pixels. Left: Uses a low threshold pair. Center: Uses a threshold pair estimated via the Otsu method. Right: Uses a high threshold pair. 89
6.7 Laplace and ZC example. Positive values of the Laplace are red, negative blue. For the ZC image on the right, the threshold was estimated with some adapted Otsu estimator for Laplace responses. 90
6.8 ZC of a disk image. The edge pixels are colored by a 4 region map (left image using fast4 zc) and an 8 region map (right image using fast8 zc). 90
6.9 Visualization of zero crossings in the Laplacian response. Red cells represent positive values, blue cells represent negative values, and a selected zero-crossing pixel is marked as green circle with its gradient direction. The dashed lines indicate interpolation lines while the filled dots represent the interpolated zero crossing position. Dark red: Linear interpolation along x-axis. Dark blue: Linear interpolation along y-axis. Dark green: Bilinear interpolation along gradient. . . . 93
6.10 Threshold examples using the Otsu threshold estimator (see subsection 6.1.4). 95
6.11 Downscaled ground truth object, ground truth line segments, Sobel 3x3 edge response and Laplace 3x3 edge response. 96
6.12 This chart shows the error results for all variants tested. For each selected category, the error is shown for the best performing variant with the lowest error. 103

6.13 This graph shows the runtime for all variants tested. For each selected category, the runtime is shown for the best performing variant with the lowest error. 104
6.14 This figure shows the upscaled polygon lines from the ground truth and the reconstructed polygon used to generate the ground truth image. It shows, as indicated in the upper image, the lower right corner of the ground truth object upscaled by a factor of 500. Color coding: gt as green, threshold as red, nms as blue, zc as yellow, nms fast spe as cyan, zc fast spe as dark orange, S9 nms prec lin spe cub dir est quad as dark blue, and L9 zc fast spe cub dir as orange. 105

7.1 This is how the edge drawing algorithm of [211] works. (a) Shows an edge map with an example edge drawing path. It starts at a seed point (local maxima, marked with a red circle, in this case the maxima at 230) and follows the edge response in the direction of the gradient (see the red and yellow circular markers). The arrows show the options depending on the horizontal (b) or vertical (c) search rule. The blue arrow indicates the selected option. From the starting seed, the edge is drawn to the left and to the right. 108
7.2 Examples for edge methods. Left: Edge Drawing. Center: Edge Linking, Right: Edge Pattern Linking. The images show the results of the non-maximum suppression (NMS) step processed with Otsu estimated thresholds. The coloring is chosen at random to distinguish the segments. 109
7.3 Examples for the edge linking method. On the left side the basic variant is shown while on the right side the corner rule is activated. 111
7.4 Examples of corner split rules. The red color indicates the current pixel in the segmentation process, the green color indicates already visited pixels, and the blue color indicates upcoming pixels. 111
7.5 Example of primitives on the left and patterns composed of primitives on the right. 112
7.6 Examples of contrast-based NFA applied to the Edge Linking method. On the left, a large NFA of 100 is selected. In the middle is the default case with an NFA of 1. On the right, an NFA of 0.001 was used. The colored edge segments passed the NFA test, all others (white) failed. 114
7.7 NFA threshold effect on Edge Linking (NFA 100, 1, 0.001). 117
7.8 NFA-sorted Edge Linking: Top 25, Top 50, Top 100 most significant segments. 118
7.9 This chart shows the runtime behavior for the ESD methods in milliseconds. CR stands for Corner Rule. 119
7.10 Examples for all edge methods. Top left to right: Edge Simple, Edge Linking and Edge Pattern Linking. Bottom left to right: Edge Drawing, Edge Linking with corner rule, Edge Pattern Linking with corner rule. The images show the results of the non-maximum suppression (NMS) step processed with Otsu estimated thresholds. The coloring is chosen at random to distinguish the segments. 120
7.11 Examples for Edge Linking with proper (Otsu) threshold on the left, Edge Linking with NFA 1 (1 average false alarm selected) in the center and Edge Linking with NFA 1 and Otsu threshold on the right. In this case the Edge Linking method also used a 10 pixel threshold to remove "unimportant" segments. This filter was not applied in the NFA cases. 122

8.1 Example of a Hough transform. The left image shows the edge pixels and the detected lines. The right image shows the Hough transform of the given edge pixels. 126
8.2 Example of over-merging. Left - Too large bins, region is merged. Right - Smaller bins, region is not merged. 129
8.3 Example for fragmentation. Left - Gradient orientations lie across a bin boundary, Right - Gradient orientations are within the bin range. 129
8.4 Example of 8 directional bins with two cases: Left - Example of a bin with 50% overlap with two other bins. The colored area shows the corresponding edge responses that fall within the bin range (all points with an orientation within the bin range are selected), Right - Example of a normal bin with corresponding edge responses. 130
8.5 This figure illustrates how the LSD algorithm works. (a) shows the extraction of level lines from the gradient image. (b) A highly magnified example of a grayscale image. (c) shows the extracted level lines based on the image gradient. (d) shows the line support regions separated by color. 131
8.6 This graphic illustrates the steps of the Ramer or Douglas-Peucker algorithm, which simplifies a curve by approximating it with linear segments. The process begins by connecting the first and last points, then recursively identifies the key points that deviate the most from the straight-line approximation. The final result is a reduced set of line segments that closely represent the original curve while minimizing complexity. 134
8.7 (a) Split rule for hard corners, (b) Split rule for curved contours. . . 135
8.8 This figure shows two merge examples of edge pattern with small and large error. Depending on a given threshold, the patterns are merged into a single pattern or split again and stored as two patterns. 137
8.9 This figure illustrates a linear regression model where a best-fit line is drawn through a set of data points. The line represents the relationship between the independent and dependent variables, minimizing the total error (residuals) between the actual points and the predicted values. The goal of linear regression is to find the optimal slope and intercept that best explain the trend in the data. 138

9.1 Point feature demo including three different point feature descriptors: AKAZE, ORB and DAISY. The image shows the detected features (left) and the matching features (right) between two images of the same scene. 150

10.1 Class diagram of the filter architecture. 163
10.2 Overview of available edge tools. 164
10.3 Edge source inheritance hierarchy. The lower part illustrates the template parameter options for ERF and EPE. 165
10.4 Edge segment inheritance and implementation hierarchy. 166
10.5 Line segment inheritance and implementation hierarchy down to LsdBase. The classes in the figure contain some template parameters that are not described in the two template argument lists. However, they refer to the edge tool abbreviations listed in the previous section. 167

10.6 Line segment inheritance and implementation hierarchy of the extended interfaces LsdCCBase and LsdExt. The classes in the figure contain some template parameters that are not described in the two template argument lists. However, they refer to the edge tool abbreviations listed in the previous section. 168
10.7 Screenshot of the Analyzer application showing multiple regions with different tasks: 1) Select image and line segment detector along with load options. 2) Line segment detector configuration. 3) Line control setup. 4) Left: list of detected line segments; Right: Controls to modify the selected line segment. 5) Image visualization options. 6) Line segment visualization options. 7) Available tools. 170
10.8 Options and dialogs: (A): Image load option dialog. (B): Image source selector for viewer. (C): Line segment detector method selector for processing. (D): Image visualization selector. 171
10.9 Analyzer Viewer to visualize images and line segments: (A): Loaded image; (B): Processed image showing the detected line segments; (C): Image source of edge segment map; (D): Image with polar visualization mode. 172
10.10 Profile Analyzer Dialog (A) and Visualizer (B). 173
10.11 Profile Analyzer Example. (A) Visualization of the line segment and profile range as an overlay on the magnitude response image. (B) Settings dialog. (C) Visualization of mean and standard deviation (blue) of the data source along the selected line segment and in the current profile at the selected profile position (red). (D) Nearest (round) interpolation, showing the "pixel" ranges for each magnitude value in the response image. 174
10.12 Precision Optimizer and PO Function Plot tools. (A) Precision Optimizer dialog. (B) PO Function Plot dialog. (C) PO Function Plot visualization showing the average magnitude calculated from the support region of the selected line segment related to the rotation angle ("Angle" axis, degree) regarding line segment center and orthogonal translation ("Profile" axis). 176
10.13 Optimization example: Left: Shows a line segment that has been manipulated by the user and is no longer perfectly aligned with the edge. Right: Shows the optimized, perfectly aligned line segment. . 178
10.14 Large range 3D plot example. This is the same local maximum as in the other 3D plots. But in this case a rotation range of $[-20, 20]$ and a profile range of $[-30, 30]$ was configured. 179
10.15 Quiver tool dialog (A) and example output (B). 180
10.16 Evaluation framework task hierarchy (ITask, Task, InputTask, MeasureTask). 183
10.17 Evaluation framework task runner hierarchy. 184

11.1 Traditional autonomous vehicle software pipeline with hierarchical layout. 195
11.2 Latest AI-based autonomous vehicle software pipeline. 195

A.1 This figure shows the even responses of the steerable quadrature filter of a simple disk image. Each row represents a different kernel size of the operator (3x3, 5x5, 7x7, and 9x9). The first column shows the filter responses for a fixed kernel spacing of 1.2. The second column shows the filter responses with the optimized kernel spacing, and the last column shows the optimized filter responses with additional zero DC correction. 199
A.2 Phase example: Top left: Example disk image; Top right: Angle color coding used for phase; Bottom left: Phase weighted by energy response of the disk image via SQFF; Bottom right: Phase via SQF 9x9 also weighted by energy response. 200
A.3 Illustrates the results of the phase differences (bars) and the DC correction values (line) that were applied to the convolutional filter. 201

List of Tables

3.1 This table shows the parameter settings for the different test cases and the resulting average information gain for each test case based on Canny. 22
3.2 This table shows the parameter settings for the different test cases and the resulting average information gain for each test case based on RCMG. 23
3.3 Includes the average absolute error after denoising and the average processing time. Five different noise levels were used (column N10-N50). 24

5.1 Execution time (ms) for gradient based edge detectors: Row "Roberts" to "Gauss" are first derivative based gradients, QF StG are quadrature based gradients (steerable filters), SQF PO are spherical quadrature filter gradients with Poisson kernels, SQFF LG and SQFF PO are spherical quadrature filters using log Gabor (LG) and Poisson (PO) kernels in the frequency domain, PC and PC ML are phase congruency gradients, Susan is a statistical gradient-like method, and RMG and RCMG are based on morphological gradients. 54
5.2 Execution time (ms) for Laplace-based edge detectors: The Laplace SQF and SQFF methods are based on phase derivatives as introduced in subsection 4.2.5. SQF is computed in the spatial domain, while SQFF is computed in the frequency domain. 55
5.3 Average execution time (ms) over data sets, based on the OpenCV Gaussian Blur operation. It includes three main variants: CPU, OCL (OpenCL) and CUDA based. The NT (no transfer) sub-variants (OCL NT and CUDA NT) show the timing without the transfer overhead of the image data to and from the GPU. 59
5.4 Average execution time (ms) over data set, based on OpenCV FFT operation. There are three main variants: CPU, OCL (OpenCL), and CUDA based. The NT (no transfer) sub-variants (OCL NT and CUDA NT) show the timing without the transfer overhead of image data to and from the GPU. 61
5.5 Average absolute differences between original image and noisy images (noise 10, 20, 30, 40 and 50) 68
5.6 The values represent the mean gradient orientation error in radians for the box setup. Each column represents a different test setting: none: original image; blur: Gaussian smoothing with $\sigma = 2$; noise 10/40: Gaussian noise with an intensity of 10/40; b+n 10/40: Gaussian smoothing and noise. 75

5.7 The values represent the average gradient orientation error in radians for the disk setup. Each column represents a different test setting: none: original image; blur: Gaussian smoothing with $\sigma = 2$; noise 10/40: Gaussian noise with an intensity of 10/40; b+n 10/40: Gaussian smoothing and noise. 76

6.1 Runtime for various thresholding methods. As expected, the more complex methods (local with tiles or dynamic) require more runtime. All methods are optimized for multi-threaded execution. The single-threaded runtimes are marked accordingly. 94
6.2 This table compares the three main edge point extraction approaches: Threshold with morphological thinning, non-maxima suppression, and zero crossing (both at the pixel level only, denoted by "nearest", which stands for nearest neighbor). The table has 6 rows, starting with the methods. EP is the number of detected edge points after the edge extraction methods, such as nms, have been applied to the Sobel or Laplace edge responses. It shows that depending on the method, a different number of edge points are collected. OL stands for outliers. It counts all edge points that could not be uniquely assigned to an edge. The Error column shows the average distance in pixels from the detected edge points to the assigned ground truth line. Edge points that fall in regions near corners are not included. The StdDev column shows the standard deviation of the error. The last row shows the runtime in milliseconds. 97
6.3 Shows fast non-maxima suppression (fast nms) with all variants for pixel interpolation and estimation for a Sobel edge response with kernel size 3x3. (Sub)pixel estimation (spe) using nearest neighbor (near), in this case no subpixel optimization. The spe linear (lin) mode uses bilinear pixel interpolation, while spe cub (cubic) uses bicubic interpolation. The dir mode indicates that directional derivatives are used instead of a simple orientation map (direction is split into 4 or 8 directional regions and stored as a lookup table). The subpixel estimation method is indicated by lin (linear), quad (quadratic), cog (center of gravity), and sob (Sobel). 98
6.4 This table shows a comparison of all main nms variants (fast, precise linear, and precise cubic), but excluding the cog and sob estimators and the spe cub variants. 99
6.5 This table shows the results for the zero crossing edge point extraction methods based on a Laplacian with a 3x3 kernel. All variants with "dir" also used a direction map computed with the Sobel operator. . 100
6.6 This table shows some detailed information about the error estimation of extracted edges by the zero crossing variants based on a 3x3 Laplacian edge response. The mean error and standard deviation are given for the first two edges. All variants giving duplicate results are removed. 101
6.7 Overview of results from multiple Sobel and Laplacian setups with kernel sizes from 3x3 to 9x9. Only the best sub-variant for each main variant is shown. A threshold only setup is also included. 102

7.1 Runtime for the ESD variants in milliseconds. CR stands for Corner Rule. 119

7.2 Runtime for the NFA variants in milliseconds. 121

8.1 Runtime for line fit methods in milliseconds. 144
8.2 Runtime for segment split methods in milliseconds. 145
8.3 Runtime for line segment detector methods in milliseconds. 146

A.1 Kernel spacing parameters estimated with the optimization method. 198
A.2 Remaining error (average of kernel) with optimal kernel spacing. . . 198

List of Abbreviations

AR	**A**ugmented **R**eality
BFGS	**B**royden **F**letcher **G**oldfarb **S**hanno
CMG	**C**olor **M**orphological **G**radient
CoG	**C**enter **o**f **G**ravity
DC	**D**irect **C**urrent
DoG	**D**ifference **o**f **G**aussians
ESD	**E**dge **S**egment **D**etector
FFT	**F**ast **F**ourier **T**ransform
FIR	**F**inite **I**mpulse **R**esponse
FT	**F**ourier **T**ransform
GPU	**G**raphic **P**rocessing **U**int
GUI	**G**raphical **U**ser **I**nterface
IIR	**I**nfinite **I**mpulse **R**esponse
LBD	**L**ine **B**and **D**escriptor
LFD	**L**ine **F**eature **D**escriptor
LG	**L**og **G**abor
LGN	**L**ateral **G**eniculate **N**ucleus
LoG	**L**aplacian **o**f **G**aussians
LSD	**L**ine **S**egment **D**etector
NFA	**N**umber of **F**alse **A**larms
NMS	**N**on **M**aximum **S**uppression
MDB	**M**iddlebury stereo evaluation **D**ata **B**ase
PCA	**P**rincipal **C**omponent **A**nalysis
PC	**P**hase **C**ongruency
PO	**PO**ssion (kernel)
QF	**Q**uadrature **F**ilter
RANSAC	**Ran**dom **Sa**mple **C**onsensus
RCMG	**R**obust **C**olor **M**orphological **G**radient
SIMD	**S**ingle **I**nstruction **M**ultiple **D**ata
SfM	**S**tructure **f**rom **M**otion
SPE	**S**ub **P**ixel **E**stimator
SQF	**S**pherical **Q**uadrature **F**ilter
SQFF	**S**pherical **Q**uadrature **F**ilter in the **F**requency domain
StG	**St**eerable **G**radient or filer
SUSAN	**S**mallest **U**nivalue **S**egment **A**ssimilation **N**ucleus
ZC	**Z**ero **C**rossing

Bibliography

[1] Radouane Ait Jellal et al. "LS-ELAS: Line Segment based Efficient Large Scale Stereo Matching". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Singapore, May 2017.

[2] Cuneyt Akinlar and Cihan Topal. "EDLines: A Real-Time Line Segment Detector with a False Detection Control". In: *Pattern Recognition Letters* 32.13 (2011), pp. 1633–1642. DOI: 10.1016/j.patrec.2011.06.001. URL: https://www.sciencedirect.com/science/article/pii/S0167865511001977.

[3] Cuneyt Akinlar and Cihan Topal. "EDPF: A Real-Time Parameter-Free Edge Segment Detector with a False Detection Control". In: *International Journal of Pattern Recognition and Artificial Intelligence* 26.1 (2012), p. 1255002. DOI: 10.1142/S0218001412550026.

[4] Fatemeh Alamdar and MohammadReza Keyvanpour. "A New Color Feature Extraction Method Based on Dynamic Color Distribution Entropy of Neighborhoods". In: *International Journal of Computer Science Issues* 8.5 (2011), pp. 41–47. URL: https://arxiv.org/abs/1201.3337.

[5] Pablo F. Alcantarilla, Jesús Nuevo, and Adrien Bartoli. "Fast Explicit Diffusion for Accelerated Features in Nonlinear Scale Spaces". In: *Proceedings of the British Machine Vision Conference (BMVC)*. 2013. DOI: 10.5244/C.27.13.

[6] Shigeru Ando. "Image Field Categorization and Edge/Corner Detection from Gradient Covariance". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 22.2 (Feb. 2000), pp. 179–190. ISSN: 0162-8828. DOI: 10.1109/34.825756. URL: https://doi.org/10.1109/34.825756.

[7] Pablo Arbelaez et al. "Contour Detection and Hierarchical Image Segmentation". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33.5 (May 2011), pp. 898–916. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2010.161. URL: https://doi.org/10.1109/TPAMI.2010.161.

[8] Ery Arias-Castro and David L. Donoho. "Does Median Filtering Truly Preserve Edges Better Than Linear Filtering?" In: *Annals of Statistics* 37.3 (2009), pp. 1172–1206. DOI: 10.1214/08-AOS606. URL: https://www.jstor.org/stable/30243665.

[9] Tetsuo Asano and Naoki Katoh. "Variants for the Hough transform for line detection". In: *Computational Geometry* 6.4 (July 1996), pp. 231–252. DOI: 10.1016/0925-7721(95)00023-2. URL: https://doi.org/10.1016/0925-7721(95)00023-2.

[10] Donald G. Bailey. "Sub-pixel Estimation of Local Extrema". In: *Proceedings of the Image and Vision Computing New Zealand Conference*. 2003, pp. 408–413.

[11] Mayank Bansal, Alex Krizhevsky, and Abhijit S. Ogale. "ChauffeurNet: Learning to Drive by Imitating the Best and Synthesizing the Worst". In: *arXiv preprint* (2018). arXiv: 1812.05890 [cs.CV].

[12] Mauro Barni and Roberto Gualtieri. "A new possibilistic clustering algorithm for line detection in real world imagery". In: *Pattern Recognition* 32.11 (1999), pp. 1897–1909. DOI: 10.1016/S0031-3203(99)00030-0. URL: https://doi.org/10.1016/S0031-3203(99)00030-0.

[13] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. "SURF: Speeded Up Robust Features". In: *Proceedings of the 9th European Conference on Computer Vision (ECCV)*. 2006, pp. 404–417. DOI: 10.1007/11744023_32. URL: https://link.springer.com/chapter/10.1007/11744023_32.

[14] G. Bellettini et al. "Some Anisotropic Diffusions". In: *Mathematics of Computation* 77.263 (2008), pp. 1175–1198. DOI: 10.1090/S0025-5718-08-02088-5. URL: https://www.ams.org/journals/mcom/2008-77-263/S0025-5718-08-02088-5/.

[15] D. Ben-Tzvi and Mark B. Sandler. "A Combinatorial Hough Transform". In: *Pattern Recognition Letters* 11.3 (1990), pp. 167–174. DOI: 10.1016/0167-8655(90)90002-J. URL: https://doi.org/10.1016/0167-8655(90)90002-J.

[16] A. Benoit et al. "Using Human Visual System Modeling for Bio-Inspired Low Level Image Processing". In: *Comput. Vis. Image Underst.* 114.7 (July 2010), pp. 758–773. ISSN: 1077-3142. DOI: 10.1016/j.cviu.2010.01.011. URL: https://doi.org/10.1016/j.cviu.2010.01.011.

[17] Nitin Bhatia and Vandana. "Survey of Nearest Neighbor Techniques". In: *International Journal of Computer Science and Information Security* 8.2 (2010), pp. 302–305. URL: https://arxiv.org/abs/1007.0085.

[18] Thierry Blaszka and Rachid Deriche. "Recovering and Characterizing Image Features Using an Efficient Model-Based Approach". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 1993, pp. 530–533.

[19] J.A. Bondy and U.S.R Murty. *Graph Theory*. 1st. Springer Publishing Company, Incorporated, 2008. ISBN: 1846289696.

[20] Jean-Yves Bouguet. *Pyramidal Implementation of the Lucas Kanade Feature Tracker: Description of the Algorithm*. Tech. rep. Intel Corporation Microprocessor Research Labs, 2001. URL: https://robots.stanford.edu/cs223b04/algo_tracking.pdf.

[21] Djamal Boukerroui, J. Alison Noble, and Michael Brady. "On the Choice of Band-Pass Quadrature Filters." In: *J. Math. Imaging Vis.* 21.1-2 (2004), pp. 53–80. URL: http://dblp.uni-trier.de/db/journals/jmiv/jmiv21.html#BoukerrouiNB04.

[22] E. Bourennane et al. "Generalization of Canny-Deriche filter for detection of noisy exponential edge". In: *Signal Processing* 82.10 (Oct. 2002), pp. 1317–1328.

[23] J.F. Boyce, J. Feng, and E.R. Haddow. "Relaxation labelling and the entropy of neighbourhood information". In: *Pattern Recognition Letters* 6.4 (1987), pp. 225–234. ISSN: 0167-8655. DOI: https://doi.org/10.1016/0167-8655(87)90081-X. URL: http://www.sciencedirect.com/science/article/pii/016786558790081X.

[24] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge, UK: Cambridge University Press, 2004. ISBN: 978-0-521-83378-3. URL: http://web.stanford.edu/~boyd/cvxbook/.

[25] Gert-Jan van den Braak et al. "Fast Hough Transform on GPUs: Exploration of Algorithm Trade-Offs". In: *Advanced Concepts for Intelligent Vision Systems (ACIVS)*. Springer. 2011, pp. 611–622. DOI: 10.1007/978-3-642-23687-7_55. URL: https://doi.org/10.1007/978-3-642-23687-7_55.

[26] Gary Bradski. "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools* (2000). URL: https://opencv.org/.

[27] Antoni Buades, Bartomeu Coll, and Jean-Michel Morel. "A Non-Local Algorithm for Image Denoising". In: *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. Vol. 2. IEEE, 2005, pp. 60–65. DOI: 10.1109/CVPR.2005.38. URL: https://ieeexplore.ieee.org/document/1467423.

[28] Antoni Buades, Bartomeu Coll, and Jean-Michel Morel. "A Review of Image Denoising Algorithms, with a New One". In: *Multiscale Modeling & Simulation* 4.2 (2005), pp. 490–530. DOI: 10.1137/040616024. URL: https://epubs.siam.org/doi/10.1137/040616024.

[29] J. B. Burns, A. R. Hanson, and E. M. Riseman. "Extracting Straight Lines". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-8.4 (1986), pp. 425–455. DOI: 10.1109/TPAMI.1986.4767808. URL: https://doi.org/10.1109/TPAMI.1986.4767808.

[30] Michael Calonder et al. "BRIEF: Binary Robust Independent Elementary Features". In: *Proceedings of the 11th European Conference on Computer Vision (ECCV)*. Springer, 2010, pp. 778–792. DOI: 10.1007/978-3-642-15561-1_56.

[31] John Canny. "A Computational Approach to Edge Detection". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 8.6 (June 1986), pp. 679–698. ISSN: 0162-8828. DOI: 10.1109/TPAMI.1986.4767851. URL: http://dx.doi.org/10.1109/TPAMI.1986.4767851.

[32] Gustavo Carneiro and Allan Jepson. "Phase-Based Local Features". In: vol. 2350. Apr. 2002, pp. 282–296. ISBN: 978-3-540-43745-1. DOI: 10.1007/3-540-47969-4_19.

[33] Thierry Carron and Philippe Lambert. "Color Edge Detector Using Jointly Hue, Saturation, and Intensity". In: *Proceedings of the 1994 IEEE International Conference on Image Processing*. Vol. 3. IEEE, 1994, pp. 977–981. DOI: 10.1109/ICIP.1994.413634. URL: https://ieeexplore.ieee.org/document/413634.

[34] Gang Chen and Yee H. Hong Yang. "Edge detection by regularized cubic B-spline fitting." In: *IEEE Trans. Syst. Man Cybern.* 25.4 (1995), pp. 636–643. URL: http://dblp.uni-trier.de/db/journals/tsmc/tsmc25.html#ChenY95.

[35] Min Chen and Zhenfeng Shao. "Robust Affine-Invariant Line Matching for High Resolution Remote Sensing Images". In: *Photogrammetric Engineering & Remote Sensing* 79.8 (2013), pp. 753–760. DOI: 10.14358/PERS.79.8.753. URL: https://www.ingentaconnect.com/content/asprs/pers/2013/00000079/00000008/art00005.

[36] E ren Chuang and David Sher. "chi2 test for feature detection." In: *Pattern Recognit.* 26.11 (1993), pp. 1671–1681. URL: http://dblp.uni-trier.de/db/journals/pr/pr26.html#ChuangS93.

[37] L. D. Cohen and I. Cohen. "Finite-element methods for active contour models and balloons for 2-D and 3-D images". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 15.11 (1993), pp. 1131–1147. DOI: 10.1109/34.244675.

[38] Gabor D. "Theory of communication. Part 1: The analysis of information". In: *Electrical Engineers - Part III: Radio and Communication Engineering, Journal of the Institution of* (1946), pp. 429–441.

[39] Rachid Deriche. "Using Canny's criteria to derive a recursively implemented optimal edge detector." In: *Int. J. Comput. Vis.* 1.2 (1987), pp. 167–187. URL: http://dblp.uni-trier.de/db/journals/ijcv/ijcv1.html#Deriche87.

[40] Agnès Desolneux, Lionel Moisan, and Jean-Michel Morel. "Edge Detection by Helmholtz Principle". In: *J. Math. Imaging Vis.* 14.3 (May 2001), pp. 271–284. ISSN: 0924-9907. DOI: 10.1023/A:1011290230196. URL: https://doi.org/10.1023/A:1011290230196.

[41] Agnès Desolneux, Lionel Moisan, and Jean-Michel Morel. *From Gestalt Theory to Image Analysis: A Probabilistic Approach*. Springer, 2008. ISBN: 978-0-387-74377-6. DOI: 10.1007/978-0-387-74378-3.

[42] Frédéric Devernay. *A Non-Maxima Suppression Method for Edge Detection with Sub-Pixel Accuracy*. Tech. rep. INRIA, 1995. URL: https://www.researchgate.net/publication/2639210_A_Non-Maxima_Suppression_Method_for_Edge_Detection_with_Sub-Pixel_Accuracy.

[43] Silvano Di Zenzo. "A Note on the Gradient of a Multi-Image". In: *Computer Vision, Graphics, and Image Processing* 33.1 (1986), pp. 116–125. DOI: 10.1016/0734-189X(86)90223-9. URL: https://doi.org/10.1016/0734-189X(86)90223-9.

[44] Lijun Ding and Alireza Goshtasby. "On the Canny Edge Detector". In: *Pattern Recognition* 34.3 (2001), pp. 721–725. DOI: 10.1016/S0031-3203(00)00023-6. URL: https://www.sciencedirect.com/science/article/pii/S0031320300000236.

[45] David H. Douglas and Thomas K. Peucker. "Algorithms for the Reduction of the Number of Points Required to Represent a Digitized Line or Its Caricature". In: *The Canadian Cartographer* 10.2 (1973), pp. 112–122. DOI: 10.3138/FM57-6770-U75U-7727. URL: https://utpjournals.press/doi/10.3138/FM57-6770-U75U-7727.

[46] Norman R. Draper and Harry Smith. *Applied Regression Analysis*. 3rd. Wiley Series in Probability and Statistics. John Wiley & Sons, 1998. ISBN: 978-0-471-17082-2. DOI: 10.1002/9781118625590. URL: https://onlinelibrary.wiley.com/doi/book/10.1002/9781118625590.

[47] Richard O. Duda and Peter E. Hart. "Use of the Hough Transformation to Detect Lines and Curves in Pictures". In: *Communications of the ACM* 15.1 (Jan. 1972), pp. 11–15. DOI: 10.1145/361237.361242. URL: https://dl.acm.org/doi/10.1145/361237.361242.

[48] James H. Elder and Richard M. Goldberg. "Ecological statistics of Gestalt laws for the perceptual organization of contours". In: *Journal of Vision* 2.4 (Aug. 2002), pp. 5–5. ISSN: 1534-7362. DOI: 10.1167/2.4.5. URL: https://doi.org/10.1167/2.4.5.

[49] E. A. Engbers and A. W. M. Smeulders. "Design considerations for generic grouping in vision". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 25.4 (2003), pp. 445–457. DOI: 10.1109/TPAMI.2003.1190571.

[50] A. N. Evans. "Morphological gradient operators for colour images". In: *2004 International Conference on Image Processing, 2004. ICIP '04.* Vol. 5. 2004, 3089–3092 Vol. 5. DOI: 10.1109/ICIP.2004.1421766.

[51] A. N. Evans and X. U. Liu. "A morphological gradient approach to color edge detection". In: *IEEE Transactions on Image Processing* 15.6 (2006), pp. 1454–1463. DOI: 10.1109/TIP.2005.864164.

[52] Bo Fan, Fuchao Wu, and Zhanyi Hu. "Robust Line Matching through Line-Point Invariants". In: *Pattern Recognition* 45.2 (2012), pp. 794–805. DOI: 10.1016/j.patcog.2011.08.001. URL: https://doi.org/10.1016/j.patcog.2011.08.001.

[53] M. Felsberg and G. Sommer. "The Monogenic Scale-Space: A Unifying Approach to Phase-Based Image Processing in Scale-Space". In: *Journal of Mathematical Imaging and Vision* 21 (2003), pp. 5–26.

[54] M. Felsberg and G. Sommer. "The monogenic signal". In: *IEEE Transactions on Signal Processing* 49.12 (2001), pp. 3136–3144. DOI: 10.1109/78.969520.

[55] Michael Felsberg. "Low-Level Image Processing with the Structure Multivector". PhD thesis. Kiel, Germany: Christian-Albrechts-Universität zu Kiel, 2002. URL: https://www.uni-kiel.de/journals/receive/jportal_jparticle_00000172.

[56] Leandro A. F. Fernandes and Manuel M. Oliveira. "Real-time line detection through an improved Hough transform voting scheme". In: *Pattern Recognition* 41.1 (2008), pp. 299–314. DOI: 10.1016/j.patcog.2007.04.003. URL: https://doi.org/10.1016/j.patcog.2007.04.003.

[57] Ondřej Fialka and Martin Čadík. "FFT and Convolution Performance in Image Filtering on GPU". In: *Proceedings of the Tenth International Conference on Information Visualisation (IV 2006)*. 2006, pp. 609–614. DOI: 10.1109/IV.2006.31. URL: https://doi.org/10.1109/IV.2006.31.

[58] David J. Field. "Relations between the statistics of natural images and the response properties of cortical cells". In: *J. Opt. Soc. Am. A* 4.12 (Dec. 1987), pp. 2379–2394. DOI: 10.1364/JOSAA.4.002379.

[59] Martin A. Fischler and Robert C. Bolles. "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography". In: *Communications of the ACM* 24.6 (June 1981), pp. 381–395. DOI: 10.1145/358669.358692. URL: https://dl.acm.org/doi/10.1145/358669.358692.

[60] Mohamad Forouzanfar and Hamid R. Rabiee. "Locally Adaptive Multiscale Bayesian Method for Image Denoising Based on Bivariate Normal Inverse Gaussian Distributions". In: *Proceedings of the 2010 IEEE International Conference on Image Processing*. IEEE, 2010, pp. 3177–3180. DOI: 10.1109/ICIP.2010.5651240. URL: https://ieeexplore.ieee.org/document/5651240.

[61] D. A. Forsyth and J. Ponce. *Computer Vision: A Modern Approach*. Prentice Hall, 2003. Chap. 8. URL: http://www.cs.berkeley.edu/~daf/book.html.

[62] William T. Freeman and Edward H. Adelson. "The Design and Use of Steerable Filters". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 13.9 (1991), pp. 891–906. DOI: 10.1109/34.93808. URL: https://ieeexplore.ieee.org/document/93808.

[63] Andreas Geiger, Martin Roser, and Raquel Urtasun. "Efficient Large-Scale Stereo Matching". In: *Asian Conference on Computer Vision (ACCV)*. 2010. DOI: 10.1007/978-3-642-19315-6_3.

[64] S. Ghosal and R. Mehrotra. "Orthogonal Moment Operators for Subpixel Edge Detection". In: *Pattern Recognition* 26.2 (1993), pp. 295–306. DOI: 10.1016/0031-3203(93)90073-7. URL: https://doi.org/10.1016/0031-3203(93)90073-7.

[65] Kuntal Ghosh, Sandip Sarkar, and Kamales Bhaumik. "A possible explanation of the low-level brightness–contrast illusions in the light of an extended classical receptive field model of retinal ganglion cells". In: *Biological Cybernetics* 94.2 (Feb. 2006), pp. 89–96. ISSN: 1432-0770. DOI: 10.1007/s00422-005-0038-4. URL: https://doi.org/10.1007/s00422-005-0038-4.

[66] Kuntal Ghosh, Sandip Sarkar, and Kamales Bhaumik. "The Theory of Edge Detection and Low-level Vision in Retrospect". In: June 2007. ISBN: 978-3-902613-05-9. DOI: 10.5772/4968.

[67] Rafael Grompone von Gioi et al. "LSD: a Line Segment Detector". In: *Image Processing On Line* 2 (2012), pp. 35–55. DOI: 10.5201/ipol.2012.gjmr-lsd. URL: https://doi.org/10.5201/ipol.2012.gjmr-lsd.

[68] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*. 2nd. Addison-Wesley Publishing Company, 1992, pp. 518–548.

[69] P. H. Gregson. "Using Angular Dispersion of Gradient Direction for Detecting Edge Ribbons." In: *IEEE Trans. Pattern Anal. Mach. Intell.* 15.7 (1993), pp. 682–696. URL: http://dblp.uni-trier.de/db/journals/pami/pami15.html#Gregson93.

[70] Cosmin Grigorescu, Nicolai Petkov, and Michel Westenberg. "Contour and boundary detection improved by surround suppression of texture edges". In: *Image and Vision Computing* 22 (Aug. 2004), pp. 609–622. DOI: 10.1016/j.imavis.2003.12.004.

[71] Cosmin Grigorescu, Nicolai Petkov, and Michel Westenberg. "Contour detection based on nonclassical receptive field inhibition". In: *IEEE transactions on image processing : a publication of the IEEE Signal Processing Society* 12 (Feb. 2003), pp. 729–39. DOI: 10.1109/TIP.2003.814250.

[72] Herbert Gross, Fritz Blechinger, and Bertram Achtner. "Human Eye". In: *Handbook of Optical Systems*. John Wiley and Sons, Ltd, 2008. Chap. 36, pp. 1–87. ISBN: 9783527699247. DOI: https://doi.org/10.1002/9783527699247.ch1. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9783527699247.ch1. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9783527699247.ch1.

[73] Stanley I. Grossman. *Elementary Linear Algebra*. Wadsworth Publishing Company, 1980. ISBN: 978-0534007461. URL: https://books.google.com/books/about/Elementary_Linear_Algebra.html?id=sz3YAAAAMAAJ.

[74] Stanley I. Grossman. *Elementary Linear Algebra*. Fifth. Harcourt College Publishers, 1994. Chap. 5. ISBN: 0-03-097354-6.

[75] Gaël Guennebaud, Benoît Jacob, et al. *Eigen: A C++ linear algebra library*. Version latest, accessed on 21 Feb 2025. 2025. URL: https://eigen.tuxfamily.org/.

[76] Rui F. C. Guerreiro and Pedro M. Q. Aguiar. "Connectivity-Enforcing Hough Transform for the Robust Extraction of Line Segments". In: *CoRR* abs/1109.3637 (2011). URL: https://arxiv.org/abs/1109.3637.

[77] Tiago H. Guerreiro, Pedro M. Q. Aguiar, and Mário A. T. Figueiredo. "Connectivity-Enforcing Hough Transform for the Robust Extraction of Line Segments". In: *IEEE Transactions on Image Processing* 21.4 (2012), pp. 2415–2428. DOI: 10.1109/TIP.2011.2181529. URL: https://ieeexplore.ieee.org/document/6129460.

[78] Teddy Gunawan et al. "Artificial Neural Network Based Fast Edge Detection Algorithm for MRI Medical Images". In: *Indonesian Journal of Electrical Engineering and Computer Science* 7 (July 2017), pp. 123–130. DOI: 10.11591/ijeecs.v7.i1.pp123-130.

[79] S. R. Gunn and M. S. Nixon. "A robust snake implementation; a dual active contour". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 19.1 (1997), pp. 63–68. DOI: 10.1109/34.566812.

[80] Gideon Guy and Gérard Medioni. "Inferring global pereeptual contours from local features". In: *International Journal of Computer Vision* 20.1 (Oct. 1996), pp. 113–133. ISSN: 1573-1405. DOI: 10.1007/BF00144119. URL: https://doi.org/10.1007/BF00144119.

[81] Miroslav Hagara and Peter Kulla. "Edge Detection with Sub-pixel Accuracy Based on Approximation of Edge with Erf Function". In: *Radioengineering* 20.2 (2011), pp. 516–524. URL: https://www.radioeng.cz/fulltexts/2011/11_02_516_524.pdf.

[82] Chris Harris and Mike Stephens. "A Combined Corner and Edge Detector". In: *Proceedings of the 4th Alvey Vision Conference*. 1988, pp. 147–151.

[83] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. 2nd. Cambridge University Press, 2004. ISBN: 978-0-521-54051-3. DOI: 10.1017/CBO9780511811685. URL: https://doi.org/10.1017/CBO9780511811685.

[84] Richard I. Hartley. "Lines and Points in Three Views and the Trifocal Tensor". In: *International Journal of Computer Vision* 22.2 (1997), pp. 125–140. DOI: 10.1023/A:1007936012022. URL: https://link.springer.com/article/10.1023/A:1007936012022.

[85] Kaiming He, Jian Sun, and Xiaoou Tang. "Guided Image Filtering". In: *Proceedings of the 11th European Conference on Computer Vision (ECCV)*. Springer, 2010, pp. 1–14. DOI: 10.1007/978-3-642-15549-9_1. URL: https://kaiminghe.com/publications/eccv10guidedfilter.pdf.

[86] E. Hecht. *Optics*. 3th. Addison-Wesley, 1998.

[87] Jeanny Herault. *Vision: Images, Signals and Neural Networks-Models of Neural Processing in Visual Perception*. Mar. 2010. ISBN: 978-981-4273-68-8. DOI: 10.1142/7311.

[88] Edwin Hewitt and Robert E. Hewitt. "The Gibbs-Wilbraham phenomenon: An episode in fourier analysis". In: *Archive for History of Exact Sciences* 21.2 (June 1979), pp. 129–160. ISSN: 1432-0657. DOI: 10.1007/BF00330404. URL: https://doi.org/10.1007/BF00330404.

[89] Takeyuki Hida et al. *White Noise: An Infinite Dimensional Calculus*. Springer Netherlands, 1993. ISBN: 978-94-017-3680-0. DOI: 10.1007/978-94-017-3680-0. URL: https://link.springer.com/book/10.1007/978-94-017-3680-0.

[90] Shusuke Hirose and Hiroshi Saito. "Fast Line Description for Line-based SLAM". In: *Proceedings of the 21st International Conference on Pattern Recognition (ICPR)*. 2012, pp. 3569–3572. DOI: 10.1109/ICPR.2012.868. URL: https://ieeexplore.ieee.org/document/6460878.

[91] Markus Hofer et al. "Efficient 3D Scene Abstraction Using Line Segments". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 4255–4264.

[92] Berthold K. P. Horn. *Robot vision*. MIT electrical engineering and computer science series. MIT Press, 1986. Chap. 3. ISBN: 978-0-262-08159-7.

[93] Paul V. C. Hough. "Method and Means for Recognizing Complex Patterns". 3,069,654. Dec. 1962. URL: https://patentimages.storage.googleapis.com/3e/8b/8b/6c4b5e9b7c8a3a/US3069654.pdf.

[94] Jiahui Huang et al. "Indoor Structure Analysis for Line-based 3D Reconstruction Using Deep Learning". In: *IEEE Transactions on Image Processing* 29 (2020), pp. 8109–8121.

[95] Jun S. Huang and Dong H. Tseng. "Statistical theory of edge detection." In: *Computer Vision, Graphics, and Image Processing* 43.3 (1988), pp. 337–346. URL: http://dblp.uni-trier.de/db/journals/cvgip/cvgip43.html#HuangT88.

[96] T. S. Huang, G. J. Yang, and G. Y. Tang. "A Fast Two-Dimensional Median Filtering Algorithm". In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 27.1 (1979), pp. 13–18. DOI: 10.1109/TASSP.1979.1163188. URL: https://ieeexplore.ieee.org/document/1163188/.

[97] J. Illingworth and J. Kittler. "A survey of the Hough transform". In: *Computer Vision, Graphics, and Image Processing* 44.1 (Aug. 1988), pp. 87–116. DOI: 10.1016/S0734-189X(88)80033-1. URL: https://doi.org/10.1016/S0734-189X(88)80033-1.

[98] J. Illingworth and J. Kittler. "The Adaptive Hough Transform". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-9.5 (1987), pp. 690–698. DOI: 10.1109/TPAMI.1987.4767964. URL: https://ieeexplore.ieee.org/document/4767964.

[99] Ziyu Jiang et al. "GraphLine3D: Structure-aware 3D Line Reconstruction with Graph Neural Networks". In: *Neural Networks* 140 (2021), pp. 31–44.

[100] I. T. Jolliffe. *Principal Component Analysis*. 2nd. Springer Series in Statistics. New York: Springer-Verlag, 2002. ISBN: 978-0-387-95442-4. DOI: 10.1007/b98835. URL: https://link.springer.com/book/10.1007/b98835.

[101] S.C. Kak. "The Discrete Hilbert Transform". In: *Proceedings of the IEEE* 58.4 (Apr. 1970), pp. 585–586. ISSN: 0018-9219. DOI: 10.1109/PROC.1970.7696.

[102] Shenbagaraj Kannapiran et al. "Stereo Visual Odometry with Deep Learning-Based Point and Line Feature Matching using an Attention Graph Neural Network". In: 2023. arXiv: 2308.01125 [cs.CV]. URL: https://arxiv.org/abs/2308.01125.

[103] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. "Snakes: Active contour models". In: *International Journal of Computer Vision* 1.4 (Jan. 1988), pp. 321–331. ISSN: 1573-1405. DOI: 10.1007/BF00133570. URL: https://doi.org/10.1007/BF00133570.

[104] Christos Katrakazas et al. "Real-time motion planning methods for autonomous on-road driving: State-of-the-art and future research directions". In: *Transportation Research Part C: Emerging Technologies* 60 (2015), pp. 416–442. DOI: 10.1016/j.trc.2015.04.002.

[105] Amandeep Kaur and Chandan Singh. "Sub-Pixel Edge Detection Using Pseudo Zernike Moment". In: *International Journal of Signal Processing, Image Processing and Pattern Recognition* 4.2 (2011), pp. 107–118.

[106] Davis E. King. *Dlib C++ Library*. Version latest, accessed on 21 Feb 2025. 2025. URL: http://dlib.net/.

[107] Ulrich Kirchmaier, Simon Hawe, and Klaus Diepold. "A Line Detection and Description Algorithm Based on Swarm Intelligence". In: *Proceedings of the 16th IEEE International Conference on Image Processing (ICIP)*. 2009, pp. 2265–2268. DOI: 10.1109/ICIP.2009.5414500. URL: https://ieeexplore.ieee.org/document/5414500.

[108] Inc. Kitware. *CMake: Cross-Platform Build System*. Version latest, accessed on 21 Feb 2025. 2025. URL: https://cmake.org/.

[109] W. Köhler. *Gestalt Psychology: An Introduction to New Concepts in Modern Psychology*. Black and Gold library. Liveright, 1970. ISBN: 9780871402189. URL: https://books.google.de/books?id=-jgN7PpjR1AC.

[110] Bryan Kolb, Ian Whishaw, and G Campbell Teskey. *An introduction to brain and behavior*. en. 6th ed. New York, UK: Macmillan Learning, 2019.

[111] S. Konishi et al. "Statistical edge detection: learning and evaluating edge cues". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 25.1 (2003), pp. 57–74. DOI: 10.1109/TPAMI.2003.1159946.

[112] Danai Koutra, Shankar Parthasarathy, and Tina Eliassi-Rad. "Algorithms for Graph Similarity and Subgraph Matching". In: *arXiv preprint arXiv:1107.2512* (2011). URL: https://arxiv.org/abs/1107.2512.

[113] Peter Kovesi. "Image Features from Phase Congruency". English. In: *Videre: Journal of Computer Vision Research* 1.3 (1999), pp. 1–26.

[114] Peter Kovesi. "Phase congruency detects corners and edges". In: *In: Proc. Austrian Patt. Recogn. Soc. Conf.* 2003, pp. 309–318.

[115] Peter Kovesi. "Phase Congruency: A Low-Level Image Invariant". English. In: *Psychological Research- Psychologische Forschung* 64.2 (2000), pp. 136–148. ISSN: 0340-0727.

[116] M.K. Kundu and S.K. Pal. "Thresholding for edge detection using human psychovisual phenomena". In: *Pattern Recognition Letters* 4.6 (1986), pp. 433–441. ISSN: 0167-8655.

[117] Dmitry Lagunovsky and Sergey Ablameyko. "Straight-line-based primitive extraction in grey-scale object recognition". In: *Pattern Recognition Letters* 20.10 (1999), pp. 1005–1014. DOI: 10.1016/S0167-8655(99)00067-7. URL: https://doi.org/10.1016/S0167-8655(99)00067-7.

[118] Manuel Lange. "Visual Odometry Using Line Features and Machine Learning Enhanced Line Description". PhD thesis. Eberhard Karls Universität Tübingen, 2023. URL: https://publikationen.uni-tuebingen.de/xmlui/bitstream/10900/134999/1/Dissertation.pdf.

[119] S. Lanser and W. Eckstein. "A modification of Deriche's approach to edge detection". In: *Proceedings., 11th IAPR International Conference on Pattern Recognition. Vol. III. Conference C: Image, Speech and Signal Analysis*, 1992, pp. 633–637. DOI: 10.1109/ICPR.1992.202067.

[120] Mi-Suen Lee and Gérard Medioni. "Grouping ., -, →,[formula], into Regions, Curves, and Junctions". In: *Computer Vision and Image Understanding* 76.1 (1999), pp. 54–69. ISSN: 1077-3142. DOI: https://doi.org/10.1006/cviu.1999.0787. URL: http://www.sciencedirect.com/science/article/pii/S1077314299907877.

[121] Vincent Leemans and Marie-France Destain. "Line cluster detection using a variant of the Hough transform for culture row localisation". In: *Image and Vision Computing* 24.5 (2006), pp. 541–550. DOI: 10.1016/j.imavis.2005.09.003. URL: https://doi.org/10.1016/j.imavis.2005.09.003.

[122] Marius Leordeanu and Martial Hebert. "A Spectral Technique for Correspondence Problems Using Pairwise Constraints". In: *Proceedings of the Tenth IEEE International Conference on Computer Vision (ICCV)*. 2005, pp. 1482–1489. DOI: 10.1109/ICCV.2005.20. URL: https://ieeexplore.ieee.org/document/1544893/.

[123] Kenneth Levenberg. "A Method for the Solution of Certain Non-Linear Problems in Least Squares". In: *Quarterly of Applied Mathematics* 2.2 (1944), pp. 164–168. DOI: 10.1090/qam/10666. URL: https://doi.org/10.1090/qam/10666.

[124] Jesse Levinson et al. "Towards Fully Autonomous Driving: Systems and Algorithms". In: *2011 IEEE International Conference on Robotics and Automation (ICRA)*. 2011, pp. 285–292. DOI: 10.1109/ICRA.2011.5980401.

[125] Bingcheng Li and Jun Shen. "Two-dimensional local moment, surface fitting and their fast computation." In: *Pattern Recognit.* 27.6 (1994), pp. 785–790. URL: http://dblp.uni-trier.de/db/journals/pr/pr27.html#LiS94.

[126] Stan Li, Han Wang, and Maria Petrou. "Relaxation Labeling of Markov Random Fields". In: (Feb. 1995).

[127] Y. Li, S. Acton, and K. Ley. "Edge Detection at Junctions". In: *Proceedings of the Alvey Vision Conference*. 1989, pp. 21–25.

[128] Dong Lim and Seung Jang. "Comparison of two-sample tests for edge detection in noisy images". In: *Journal of the Royal Statistical Society: Series D (The Statistician)* 51 (May 2002), pp. 21–30. DOI: 10.1111/1467-9884.00295.

[129] Tony Lindeberg. "Edge Detection and Ridge Detection with Automatic Scale Selection". In: *International Journal of Computer Vision* 30.2 (1998), pp. 117–154. DOI: 10.1023/A:1008097225773. URL: https://link.springer.com/article/10.1023/A:1008097225773.

[130] Tony Lindeberg. *Scale-Space Theory in Computer Vision*. Springer, 1994. ISBN: 978-0-7923-9418-4. DOI: 10.1007/978-1-4757-6465-9.

[131] Ce Liu et al. "Noise Estimation from a Single Image". In: *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*. 2006, pp. 901–908. DOI: 10.1109/CVPR.2006.172. URL: https://people.csail.mit.edu/billf/publications/Noise_Estimation_Single_Image.pdf.

[132] Google LLC. *Bazel: A Fast, Scalable, Multi-Language Build System*. Version latest, accessed on 21 Feb 2025. 2025. URL: https://bazel.build/.

[133] Juan López et al. "Two-view line matching algorithm based on context and appearance in low-textured images". In: *Pattern Recognition* 48.7 (2015), pp. 2164–2184. ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2014.11.018. URL: http://www.sciencedirect.com/science/article/pii/S0031320314004889.

[134] D. G. Lowe. "Object recognition from local scale-invariant features". In: *Proceedings of the Seventh IEEE International Conference on Computer Vision*. Vol. 2. 1999, 1150–1157 vol.2. DOI: 10.1109/ICCV.1999.790410.

[135] David G. Lowe. "Distinctive Image Features from Scale-Invariant Keypoints". In: *International Journal of Computer Vision* 60.2 (2004), pp. 91–110. DOI: 10.1023/B:VISI.0000029664.99615.94. URL: https://doi.org/10.1023/B:VISI.0000029664.99615.94.

[136] Dingran Lu et al. "Neural Network Based Edge Detection for Automated Medical Diagnosis". In: (June 2011). DOI: 10.1109/ICINFA.2011.5949014.

[137] Bruce D. Lucas and Takeo Kanade. "An Iterative Image Registration Technique with an Application to Stereo Vision". In: *Proceedings of the 7th International Joint Conference on Artificial Intelligence (IJCAI)*. 1981, pp. 674–679. URL: https://www.ijcai.org/Proceedings/81-2/Papers/017.pdf.

[138] Yuancheng Luo and R. Duraiswami. "Canny edge detection on NVIDIA CUDA". In: *Computer Vision and Pattern Recognition Workshops, 2008. CVPRW '08. IEEE Computer Society Conference on*. June 2008, pp. 1–8. DOI: 10.1109/CVPRW.2008.4563088. URL: https://www.computer.org/csdl/proceedings-article/cvprw/2008/04563088/12OmNqHItxJ.

[139] Quang-Tuan Luong and Olivier D. Faugeras. "The Fundamental Matrix: Theory, Algorithms, and Stability Analysis". In: *International Journal of Computer Vision* 17.1 (1996), pp. 43–75. DOI: 10.1007/BF00127818. URL: https://link.springer.com/article/10.1007/BF00127818.

[140] Richard G. Lyons. *Understanding Digital Signal Processing (2nd Edition)*. USA: Prentice Hall PTR, 2004. Chap. 5-6. ISBN: 0131089897.

[141] E. R. Lyvers et al. "Subpixel Measurements Using a Moment-Based Edge Operator". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 11.12 (1989), pp. 1293–1309.

[142] Song De MA and Bingcheng Li. "Derivative computation by multiscale filters". In: *Image and Vision Computing* 16.1 (1998). Geometric Modeling and Invariants for Computing Vision, pp. 43–53. ISSN: 0262-8856. DOI: https://doi.org/10.1016/S0262-8856(97)00042-5. URL: http://www.sciencedirect.com/science/article/pii/S0262885697000425.

[143] Raul Machuca and K. Phillips. "Applications of Vector Fields to Image Processing". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-5.2 (1983), pp. 181–188. DOI: 10.1109/TPAMI.1983.4767393. URL: https://ieeexplore.ieee.org/document/4767393.

[144] Michael Maire et al. "Using Contours to Detect and Localize Junctions in Natural Images". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2008, pp. 1–8. DOI: 10.1109/CVPR.2008.4587583. URL: https://doi.org/10.1109/CVPR.2008.4587583.

[145] Stéphane Mallat. *A Wavelet Tour of Signal Processing*. Academic Press, 2008. ISBN: 978-0-12-374370-5. DOI: 10.1016/B978-0-12-374370-1.X0001-8.

[146] Donald W. Marquardt. "An Algorithm for Least-Squares Estimation of Nonlinear Parameters". In: *SIAM Journal on Applied Mathematics* 11.2 (1963), pp. 431–441. DOI: 10.1137/0111030. URL: https://doi.org/10.1137/0111030.

[147] D. Marr and E. Hildreth. "Theory of Edge Detection". In: *Proceedings of the Royal Society of London. Series B, Containing papers of a Biological character. Royal Society (Great Britain)* 207.1167 (Feb. 1980), pp. 187–217. ISSN: 0080-4649. DOI: 10.1098/rspb.1980.0020. eprint: http://rspb.royalsocietypublishing.org/content/207/1167/187.full.pdf. URL: http://rspb.royalsocietypublishing.org/content/207/1167/187.

[148] D. R. Martin, C. C. Fowlkes, and J. Malik. "Learning to detect natural image boundaries using local brightness, color, and texture cues". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 26.5 (2004), pp. 530–549. DOI: 10.1109/TPAMI.2004.1273918.

[149] P. Meer and B. Georgescu. "Edge detection with embedded confidence". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 23.12 (2001), pp. 1351–1365. DOI: 10.1109/34.977560.

[150] Tim S. Meese et al. "Contextual modulation involves suppression and facilitation from the center and the surround". In: *Journal of Vision* 7.4 (Mar. 2007), pp. 7–7. ISSN: 1534-7362. DOI: 10.1167/7.4.7. URL: https://doi.org/10.1167/7.4.7.

[151] Yves Meyer. *Wavelets: Algorithms and Applications*. Society for Industrial and Applied Mathematics, 1993. ISBN: 978-0-89871-309-1. DOI: 10.1137/1.9780898718119.

[152] R. Mohan and R. Nevatia. "Perceptual organization for scene segmentation and description". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 14.6 (1992), pp. 616–635. DOI: 10.1109/34.141553.

[153] M. Concetta Morrone and Robyn A. Owens. "Feature detection from local energy." In: *Pattern Recognit. Lett.* 6.5 (1987), pp. 303–313. URL: http://dblp.uni-trier.de/db/journals/prl/prl6.html#Morrone087.

[154] Beatrice Alessandra Motetti et al. "Adaptive Deep Learning for Efficient Visual Pose Estimation aboard Ultra-low-power Nano-drones". In: *arXiv preprint arXiv:2401.15236* (2024).

[155] Parimal Mukhopadhyay. *An Introduction to Estimating Functions*. Alpha Science International, Ltd., 2004. ISBN: 978-18-426-5164-5. URL: https://www.alphasci.com/bookdetails/An-Introduction-to-Estimating-Functions.

[156] Hanna Müller, Victor Kartsch, and Luca Benini. "GAP9Shield: A 150GOPS AI-capable Ultra-low Power Module for Vision and Ranging Applications on Nano-drones". In: *arXiv preprint arXiv:2407.13706* (2024).

[157] T. Mundhenk and Laurent Itti. "Computational modeling and exploration of contour integration for visual saliency". In: *Biological cybernetics* 93 (Oct. 2005), pp. 188–212. DOI: 10.1007/s00422-005-0577-8.

[158] Alexander Neubeck and Luc Van Gool. "Efficient Non-Maximum Suppression". In: *Proceedings of the 18th International Conference on Pattern Recognition (ICPR 2006)*. Vol. 3. 2006, pp. 850–855. DOI: 10.1109/ICPR.2006.479. URL: https://doi.org/10.1109/ICPR.2006.479.

[159] R. Nevatia. "A Color Edge Detector and Its Use in Scene Segmentation". In: *IEEE Transactions on Systems, Man, and Cybernetics* SMC-7.11 (1977), pp. 820–826. DOI: 10.1109/TSMC.1977.4309881. URL: https://ieeexplore.ieee.org/document/4309881.

[160] Mark Nitzberg, David Mumford, and Takahiro Shiota. *Filtering, Segmentation and Depth*. Vol. 662. Lecture Notes in Computer Science. Springer, 1993. ISBN: 3-540-56484-5.

[161] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. 2nd. Springer Series in Operations Research and Financial Engineering. Springer, 2006. ISBN: 978-0-387-30303-1. DOI: 10.1007/978-0-387-40065-5.

[162] Cynthia L. Novak and Steven A. Shafer. "Color Edge Detection". In: *DARPA Image Understanding Workshop*. 1987, pp. 35–37.

[163] A. B. J. Novikoff. "Integral Geometry as a Tool in Pattern Recognition". In: *Principles of Self-Organization*. Ed. by Heinz von Foerster and Jr. George W. Zopf. New York, NY: Pergamon, 1962, pp. 347–368.

[164] Nobuyuki Otsu. "A Threshold Selection Method from Gray-Level Histograms". In: *IEEE Transactions on Systems, Man, and Cybernetics* 9.1 (1979), pp. 62–66. DOI: 10.1109/TSMC.1979.4310076. URL: https://ieeexplore.ieee.org/document/4310076.

[165] Brian Paden et al. "A Survey of Motion Planning and Control Techniques for Self-Driving Urban Vehicles". In: *IEEE Transactions on Intelligent Vehicles* 1.1 (2016), pp. 33–55. DOI: 10.1109/TIV.2016.2578706.

[166] Daniele Palossi et al. "Fully Onboard AI-powered Human-Drone Pose Estimation on Ultra-low Power Autonomous Flying Nano-UAVs". In: *arXiv preprint arXiv:2103.10873* (2021).

[167] Giuseppe Papari and Nicolai Petkov. "Edge and line oriented contour detection: State of the art". In: *Image and Vision Computing* 29.2 (2011), pp. 79–103. ISSN: 0262-8856. DOI: https://doi.org/10.1016/j.imavis.2010.08.009. URL: http://www.sciencedirect.com/science/article/pii/S0262885610001253.

[168] Giuseppe Papari et al. "A Biologically Motivated Multiresolution Approach to Contour Detection". In: *EURASIP Journal on Advances in Signal Processing* 2007.1 (Dec. 2007), p. 071828. ISSN: 1687-6180. DOI: 10.1155/2007/71828. URL: https://doi.org/10.1155/2007/71828.

[169] P. Parent and S. W. Zucker. "Trace inference, curvature consistency, and curve detection". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 11.8 (1989), pp. 823–839. DOI: 10.1109/34.31445.

[170] Christopher Passaglia, Christina Enroth-Cugell, and John Troy. "Effects of Remote Stimulation on the Mean Firing Rate of Cat Retinal Ganglion Cells". In: *The Journal of neuroscience : the official journal of the Society for Neuroscience* 21 (Sept. 2001), pp. 5794–803. DOI: 10.1523/JNEUROSCI.21-15-05794.2001.

[171] Nicolai Petkov and Michel Westenberg. "Suppression of contour perception by band-limited noise and its relation to nonclassical receptive field inhibition". In: *Biological cybernetics* 88 (Apr. 2003), pp. 236–46. DOI: 10.1007/s00422-002-0378-2.

[172] Tomas Petricek. "Library patterns: Why frameworks are evil". In: (2013). URL: https://tomasp.net/blog/2015/library-frameworks/.

[173] Florian Pilz, Nicolas Pugeault, and Norbert Krüger. "Comparison of Point and Line Features and Their Combination for Rigid Body Motion Estimation". In: vol. 5604. July 2009, pp. 280–304. ISBN: 978-3-642-03060-4. DOI: 10.1007/978-3-642-03061-1_14.

[174] J. M. S. Prewitt. "Object enhancement and extraction". In: *Picture Processing and Psychopictorics* (1970), pp. 75–149.

[175] J. Princen, J. Illingworth, and J. Kittler. "A Hierarchical Approach to Line Extraction based on the Hough Transform". In: *Computer Vision, Graphics, and Image Processing* 52.1 (1990), pp. 57–77. DOI: 10.1016/0734-189X(90)90123-D. URL: https://dl.acm.org/doi/10.1016/0734-189X%2890%2990123-D.

[176] Dale Purves, Amita Shimpi, and R. Lotto. "An Empirical Explanation of the Cornsweet Effect". In: *The Journal of neuroscience : the official journal of the Society for Neuroscience* 19 (Nov. 1999), pp. 8542–51. DOI: 10.1523/JNEUROSCI.19-19-08542.1999.

[177] Urs Ramer. "An Iterative Procedure for the Polygonal Approximation of Plane Curves". In: *Computer Graphics and Image Processing* 1.3 (1972), pp. 244–256. DOI: 10.1016/S0146-664X(72)80017-0. URL: https://doi.org/10.1016/S0146-664X(72)80017-0.

[178] Floyd Ratliff. "Mach Bands: Quantitative Studies on Neural Networks in the Retina". In: *Science* 150.3696 (1965), pp. 596–597. ISSN: 0036-8075. DOI: 10.1126/science.150.3696.596. eprint: https://science.sciencemag.org/content/150/3696/596.full.pdf. URL: https://science.sciencemag.org/content/150/3696/596.

[179] Jean-François Rivest, Pierre Soille, and Serge Beucher. "Morphological gradients". In: *J. Electronic Imaging* 2 (Jan. 1993), pp. 326–336. DOI: 10.1117/12.159642.

[180] A. Rosenfeld, R. A. Hummel, and S. W. Zucker. "Scene Labeling by Relaxation Operations". In: *IEEE Transactions on Systems, Man, and Cybernetics* SMC-6.6 (1976), pp. 420–433. DOI: 10.1109/TSMC.1976.4309519.

[181] Azriel Rosenfeld. *Picture Processing by Computer*. Academic Press, 1969. ISBN: 978-01-259-7550-0.

[182] C. A. Rothwell et al. "Driving Vision by Topology". In: *Proceedings of the International Symposium on Computer Vision*. 1995, pp. 395–400. URL: https://www.robots.ox.ac.uk/~vgg/publications/1995/Rothwell95b/rothwell95b.pdf.

[183] Ethan Rublee et al. "ORB: An efficient alternative to SIFT or SURF". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2011, pp. 2564–2571. DOI: 10.1109/ICCV.2011.6126544. URL: https://ieeexplore.ieee.org/document/6126544.

[184] Yossi Rubner, Carlo Tomasi, and Leonidas J. Guibas. "A Metric for Distributions with Applications to Image Databases". In: *Proceedings of the Sixth International Conference on Computer Vision (ICCV)*. IEEE, 1998, pp. 59–66. DOI: 10.1109/ICCV.1998.710701. URL: https://ieeexplore.ieee.org/document/710701.

[185] L. I. Rudin, S. Osher, and E. Fatemi. "Nonlinear Total Variation Based Noise Removal Algorithms". In: *Physica D: Nonlinear Phenomena* 60.1-4 (1992), pp. 259–268. DOI: 10.1016/0167-2789(92)90242-F. URL: https://doi.org/10.1016/0167-2789(92)90242-F.

[186] M A Ruzon and C Tomasi. "Edge, Junction, and Corner Detection Using Color Distributions". In: *IEEE Trans. Pattern Analysis Machine Intelligence* 23.11 (2001), pp. 1281–1295. ISSN: 0162-8828. DOI: http://dx.doi.org/10.1109/34.969118.

[187] Mark A. Ruzon and Carlo Tomasi. "Color Edge Detection with the Compass Operator". In: *Proceedings of the 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 1999, pp. 160–166. DOI: 10.1109/CVPR.1999.784624. URL: https://ieeexplore.ieee.org/document/784624/.

[188] R.A. Salinas et al. "Data Fusion: Color Edge Detection and Surface Reconstruction Through Regularization". In: *IEEE Transactions on Industrial Electronics* 43.3 (1996), pp. 355–363. DOI: 10.1109/41.498807. URL: https://ieeexplore.ieee.org/document/498807.

[189] Mohamed El-Sayed and Muhammad Ahmed. "Automated Edge Detection Using Convolutional Neural Network". In: *International Journal of Advanced Computer Science and Applications* 4 (Jan. 2013).

[190] J. Scharcanski and A. N. Venetsanopoulos. "Edge detection of color images using directional operators". In: *IEEE Transactions on Circuits and Systems for Video Technology* 7.2 (1997), pp. 397–401. DOI: 10.1109/76.564116.

[191] Hanno Scharr. *Optimale Operatoren in der Digitalen Bildverarbeitung*. 2000. URL: http://archiv.ub.uni-heidelberg.de/volltextserver/962/.

[192] Daniel Scharstein et al. "High-Resolution Stereo Datasets with Subpixel-Accurate Ground Truth". In: *36th German Conference on Pattern Recognition (GCPR 2014)*. Vol. 8753. Lecture Notes in Computer Science. Münster, Germany: Springer, Sept. 2014, pp. 31–42. ISBN: 978-3-319-11751-5. DOI: 10.1007/978-3-319-11752-2_3. URL: https://doi.org/10.1007/978-3-319-11752-2_3.

[193] Cordelia Schmid and Andrew Zisserman. "Automatic Line Matching across Views". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*. 1997, pp. 666–671. DOI: 10.1109/CVPR.1997.609406. URL: https://doi.org/10.1109/CVPR.1997.609406.

[194] Cordelia Schmid and Andrew Zisserman. "The Geometry and Matching of Lines and Curves Over Multiple Views". In: *International Journal of Computer Vision* 40.3 (2000), pp. 199–233. DOI: 10.1023/A:1008135310502. URL: https://link.springer.com/article/10.1023/A:1008135310502.

[195] L. Shafarenko, M. Petrou, and J. Kittler. "Automatic watershed segmentation of randomly textured color images". In: *IEEE Transactions on Image Processing* 6.11 (1997), pp. 1530–1544. DOI: 10.1109/83.641413.

[196] Jun Shen and Serge Castan. "An optimal linear operator for step edge detection." In: *CVGIP: Graphical Model and Image Processing* 54.2 (1992), pp. 112–133. URL: http://dblp.uni-trier.de/db/journals/cvgip/cvgip54.html#ShenC92.

[197] Chandan Singh and Nitin Bhatia. "A Fast Decision Technique for Hierarchical Hough Transform for Line Detection". In: *CoRR* abs/1007.0547 (2010). URL: https://arxiv.org/abs/1007.0547.

[198] S M Smith and J M Brady. "SUSAN-A New Approach to Low Level Image Processing". In: *Int. Journal of Computer Vision* 23.1 (1997), pp. 45–78. ISSN: 0920-5691.

[199] Irwin Sobel and Gary Feldman. "A 3×3 isotropic gradient operator for image processing". In: *Pattern Classification and Scene Analysis*. John Wiley & Sons, 1973, pp. 271–272.

[200] Michael Spratling. "Image Segmentation Using a Sparse Coding Model of Cortical Area V1". In: *IEEE transactions on image processing : a publication of the IEEE Signal Processing Society* 22 (Dec. 2012). DOI: 10.1109/TIP.2012.2235850.

[201] J. S. Stahl and S. Wang. "Globally Optimal Grouping for Symmetric Closed Boundaries by Combining Boundary and Region Information". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 30.3 (2008), pp. 395–411. DOI: 10.1109/TPAMI.2007.1186.

[202] L. H. Staib and J. S. Duncan. "Boundary finding with parametrically deformable models". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 14.11 (1992), pp. 1061–1075. DOI: 10.1109/34.166621.

[203] George Stockman and Linda G. Shapiro. *Computer Vision*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2001. Chap. 5.6. ISBN: 0130307963.

[204] G. Strang. *Calculus*. Open Textbook Library Bd. 1. Wellesley-Cambridge Press, 1991. ISBN: 9780961408824. URL: https://books.google.de/books?id=0isInC1zvEMC.

[205] Richard Szeliski. *Computer Vision: Algorithms and Applications*. 1st. New York, NY, USA: Springer-Verlag New York, Inc., 2010. ISBN: 9781848829343.

[206] ADAPT Project Team. "Open Source Payload Project For In-Flight AI Computer Vision". In: (2021). https://discuss.px4.io/t/open-source-payload-project-for-in-flight-ai-computer-vision/24250.

[207] The Qt Company. *Qt 5*. A cross-platform application development framework. Version 5. 2012. URL: https://www.qt.io/.

[208] Engin Tola, Vincent Lepetit, and Pascal Fua. "A Fast Local Descriptor for Dense Matching". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2008, pp. 1–8. DOI: 10.1109/CVPR.2008.4587673.

[209] Carlo Tomasi and Roberto Manduchi. "Bilateral Filtering for Gray and Color Images". In: *Proceedings of the 1998 IEEE International Conference on Computer Vision*. IEEE, 1998, pp. 839–846. DOI: 10.1109/ICCV.1998.710815. URL: https://ieeexplore.ieee.org/document/710815.

[210] Cihan Topal and Cuneyt Akinlar. "Edge Drawing: A combined real-time edge and segment detector". In: *Journal of Visual Communication and Image Representation* 23.6 (2012), pp. 862–872. DOI: 10.1016/j.jvcir.2012.05.004.

[211] Cihan Topal, Cuneyt Akinlar, and Yakup Genc. "Edge Drawing: A Heuristic Approach to Robust Real-Time Edge Detection". In: *Proceedings of the 20th International Conference on Pattern Recognition (ICPR)*. 2010, pp. 2424–2427. DOI: 10.1109/ICPR.2010.593. URL: https://ieeexplore.ieee.org/document/5595747/.

[212] P.H.S. Torr and A. Zisserman. "Robust Parameterization and Computation of the Trifocal Tensor". In: *Proceedings of the British Machine Vision Conference (BMVC)*. 1996, pp. 861–870. DOI: 10.5244/C.10.61. URL: https://www.bmva.org/bmvc/1996/torr_1.html.

[213] P. E. Trahanias and A. N. Venetsanopoulos. "Color edge detection using vector order statistics". In: *IEEE Transactions on Image Processing* 2.2 (1993), pp. 259–264. DOI: 10.1109/83.217230.

[214] P. E. Trahanias and A. N. Venetsanopoulos. "Vector Order Statistics Operators as Color Edge Detectors". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 26.1 (1996), pp. 135–143. DOI: 10.1109/3477.484445. URL: https://ieeexplore.ieee.org/document/484445/.

[215] W.H. Tsang and P.W.M. Tsang. "Suppression of False Edge Detection Due to Specular Reflection in Color Images". In: *Pattern Recognition Letters* 18.2 (1997), pp. 165–171. DOI: 10.1016/S0167-8655(96)00112-0. URL: https://doi.org/10.1016/S0167-8655(96)00112-0.

[216] Yung-Fu Tsao and King-Sun Fu. "A Parallel Thinning Algorithm for 3-D Pictures". In: *Computer Graphics and Image Processing* 17.4 (1981), pp. 315–331.

[217] T. Tsenoglou, N. Vassilas, and D. Ghazanfarpour. "A New Weighted Region-based Hough Transform Algorithm for Robust Line Detection in Poor Quality Images of 2D Lattices of Rectangular Objects". In: *Electronic Letters on Computer Vision and Image Analysis* 12.2 (2013), pp. 1–13. URL: https://elcvia.cvc.uab.es/article/view/v12-n2-tsenoglou-vassilas-ghazanfarpour.

[218] Koji Tsuda, Michihiko Minoh, and Katsuo Ikeda. "Extracting straight lines by sequential fuzzy clustering". In: *Pattern Recognition Letters* 17.6 (May 1996), pp. 643–649. DOI: 10.1016/0167-8655(96)00029-3. URL: https://doi.org/10.1016/0167-8655(96)00029-3.

[219] John W. Tukey. "Nonlinear (Nonsuperposable) Methods for Smoothing Data". In: *Cong. Rec. EASCON* (1974), pp. 673–678.

[220] Mauro Ursino and Giuseppe Emiliano La Cara. "A model of contextual interactions and contour detection in primary visual cortex". In: *Neural Networks* 17.5 (2004). Vision and Brain, pp. 719–735. ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2004.03.007. URL: http://www.sciencedirect.com/science/article/pii/S0893608004000796.

[221] S. Venkatesh and R. Owens. "An energy feature detection scheme". In: *ICIP '89 : IEEE International Conference on Image Processing : conference proceedings, 5-8 September 1989, Singapore*. IEEE, 1989.

[222] Wai-Shun Tong and Chi-Keung Tang. "Robust estimation of adaptive tensors of curvature by tensor voting". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 27.3 (2005), pp. 434–449. DOI: 10.1109/TPAMI.2005.62.

[223] Michael E. Wall, Andreas Rechtsteiner, and Luis M. Rocha. "Singular Value Decomposition and Principal Component Analysis". In: *A Practical Approach to Microarray Data Analysis*. Ed. by Daniel P. Berrar, Werner Dubitzky, and Martin Granzow. Springer, 2003, pp. 91–109. DOI: 10.1007/0-306-47815-3_5. URL: https://link.springer.com/chapter/10.1007/0-306-47815-3_5.

[224] S. Wang et al. "Salient closed boundary extraction with ratio contour". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 27.4 (2005), pp. 546–561. DOI: 10.1109/TPAMI.2005.84.

[225] Zhaopeng Wang, Fuchao Wu, and Zhanyi Hu. "HLD: A Robust Descriptor for Line Matching". In: *Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 1–8. DOI: 10.1109/CVPR.2009.5206757. URL: https://ieeexplore.ieee.org/document/5206757.

[226] Zhaopeng Wang, Fuchao Wu, and Zhanyi Hu. "MSLD: A Robust Descriptor for Line Matching". In: *Pattern Recognition* 42.5 (2009), pp. 941–953. DOI: 10.1016/j.patcog.2008.08.035. URL: https://doi.org/10.1016/j.patcog.2008.08.035.

[227] Benjamin Wassermann. *Line Extraction Tool Library*. GitHub repository, accessed on February 12025. 2025. URL: https://github.com/waterben/LineExtraction.

[228] Lance R. Williams and Karvel K. Thornber. "A Comparison of Measures for Detecting Natural Shapes in Cluttered Backgrounds". In: *International Journal of Computer Vision* 34.2 (Aug. 1999), pp. 81–96. ISSN: 1573-1405. DOI: 10.1023/A:1008187804026. URL: https://doi.org/10.1023/A:1008187804026.

[229] Andrew P. Witkin. "Scale-space Filtering". In: *Proceedings of the Eighth International Joint Conference on Artificial Intelligence - Volume 2*. IJCAI'83. Karlsruhe, West Germany: Morgan Kaufmann Publishers Inc., 1983, pp. 1019–1022. URL: http://dl.acm.org/citation.cfm?id=1623516.1623607.

[230] Brandon Wozniewicz. "The Difference Between a Framework and a Library". In: *freeCodeCamp* (2018). URL: https://www.freecodecamp.org/news/the-difference-between-a-framework-and-a-library-bd133054023f/.

[231] Lei Xu, Erkki Oja, and Pekka Kultanen. "A New Curve Detection Method: Randomized Hough Transform (RHT)". In: *Pattern Recognition Letters* 11.5 (May 1990), pp. 331–338. DOI: 10.1016/0167-8655(90)90042-Z. URL: https://doi.org/10.1016/0167-8655(90)90042-Z.

[232] Sungho Yoon and Ayoung Kim. "Line as a Visual Sentence: Context-Aware Line Descriptor for Visual Localization". In: *IEEE Robotics and Automation Letters* 6.4 (Oct. 2021), pp. 8726–8733. ISSN: 2377-3774. DOI: 10.1109/lra.2021.3111760. URL: http://dx.doi.org/10.1109/LRA.2021.3111760.

[233] Ian T. Young, Jan J. Gerbrands, and Lucas J. van Vliet. *Fundamentals of Image Processing*. Version 2.3. Delft University of Technology, 2007. URL: https://www.mathworks.com/matlabcentral/mlc-downloads/downloads/submissions/25588/versions/1/previews/Fundamentals%20of%20Image%20Processing.pdf.

[234] W. Yu, G. W. Chu, and M. J. Chung. "A robust line extraction method by unsupervised line clustering". In: *Pattern Recognition* 32.4 (1999), pp. 529–546. DOI: 10.1016/S0031-3203(98)00101-0. URL: https://doi.org/10.1016/S0031-3203(98)00101-0.

[235] Fang Yue and Zhang Xuewu. "The Study on An Application of Otsu Method in Canny Operator". In: *2009 Fifth International Conference on Natural Computation*. 2009, pp. 33–37. DOI: 10.1109/ICNC.2009.239. URL: https://ieeexplore.ieee.org/document/5360853.

[236] Lilian Zhang and Reinhard Koch. "An Efficient and Robust Line Segment Matching Approach Based on LBD Descriptor and Pairwise Geometric Consistency". In: *Journal of Visual Communication and Image Representation* 24.7 (2013), pp. 794–805. DOI: 10.1016/j.jvcir.2013.05.006. URL: https://doi.org/10.1016/j.jvcir.2013.05.006.

[237] T. Y. Zhang and C. Y. Suen. "A Fast Parallel Algorithm for Thinning Digital Patterns". In: *Communications of the ACM* 27.3 (1984), pp. 236–239. DOI: 10.1145/357994.358023.

[238] Wenbin Zhou et al. "Learning Structural Priors for Indoor 3D Line Reconstruction". In: *Computer Vision and Image Understanding* 188 (2019), p. 102795.

[239] Shuyun Zhu, Konstantinos N. Plataniotis, and Anastasios N. Venetsanopoulos. "Comprehensive Analysis of Edge Detection in Color Image Processing". In: *Optical Engineering* 38.4 (1999), pp. 612–625. DOI: 10.1117/1.602125. URL: https://doi.org/10.1117/1.602125.