Analysis of Optimization Methods in Multisteerable Filter Design

Philip Zanco
University of New Orleans, New Orleans, pzanco@uno.edu

Analysis of Optimization Methods in Multisteerable Filter Design

A Thesis

Submitted to Graduate Faculty of the
University of New Orleans
in partial fulfillment of the
requirements for the degree of

Master of Science
in
Engineering
Electrical

By

Philip Ronald Zanco

B.S. University of New Orleans, 2013

August, 2016

To my parents, family, colleagues, and friends

ACKNOWLEDGEMENT

This thesis would not have been possible without the guidance and help of several individuals who in one way or another contributed and extended their valuable assistance in the preparation and completion of this study.

First and foremost, I would like to express my utmost gratitude to my advisor, Professor Dimitrios Charalampidis for his valuable guidance and selfless support during the years of my Masters studies. His patience and unfailing encouragement have been the major contributing factors in the completion of my thesis research.

I am also extremely thankful to my parents, and sister for their never ending love and support. Finally, I would like to thank all the professors and staff at the University of New Orleans who contributed in providing me with a quality education.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

The purpose of this thesis is to study and investigate a practical and efficient implementation of corner orientation detection using multisteerable filters. First, practical theory involved in applying multisteerable filters for corner orientation estimation is presented. Methods to improve the efficiency with which multisteerable corner filters are applied to images are investigated and presented. Prior research in this area presented an optimization equation for determining the best match of corner orientations in images; however, little research has been done on optimization techniques to solve this equation. Optimization techniques to find the maximum response of a similarity function to determine how similar a corner feature is to a multioriented corner template are also explored and compared in this research.

Key Words: Steerable filters, feature detection, orientation estimation, corner detection

CHAPTER 1

INTRODUCTION

Many computer vision applications involve using features to obtain some sort of useful information from images. In particularly, many features are characterized by their orientation. A feature may be characterized by a single orientation angle, or multiple orientation angles. Steerable filters provide an efficient framework to determine orientation angles of various types of features that are oriented in one or more directions.

1.1 Historical Review of Steerable Filter Applications

For this thesis, much research was done on applications of multisteerable filters. This research provided the motivation for exploring more efficient techniques to automatically determine the orientation of corner features using multisteerable filters. Some examples of important applications of steerable filters include palmprint recognition [1], image stitching [2], driver assistance systems [3], facial expression recognition [4], scene flow estimation [5], detection of cerebral vessels [6], and many more.

These computer vision applications are all based on the detection of features to gather useful information from an image. Much research has been done in the area of steerable filter design for detecting features such as edges [7], [8], [9]. However, edges are simple features that are only oriented in one direction; thus, they can be steered with only one orientation angle and can be detected by using single steerable filters. Many useful computer vision applications are based on detecting features that are oriented in multiple directions, such as crossings, wedges, and corners [9]. The development of the theory for designing steerable filters which can be steered in two or more directions was thus an important development in the field of computer vision.

Applications such as palmprint recognition use steerable filters to detect simple features such as the orientation of palm lines, and match people based on their palmprint [1]. Such applications are important for access granting and security systems, as well as criminal investigations. Steerable filters provide an efficient framework for quickly detecting these features in multispectral images, and using score level fusion techniques to determine likely matches between palm prints.

Another important application of steerable filters involves image stitching. Image stitching is the process of taking multiple images of the same scene taken from different viewpoints, and attempting to resolve their homography differences by determining features in images. Image stitching has historically been done by determining corner features in images [10]. There has been much research and progress on the detection of corner features [11]; however, little research has been done on efficiently detecting the orientation of corners. Moreover, accurately determining the orientation of corners in an image requires the detection of two orientation angles since a corner is made up of two edges oriented at different angles [9]. Historically, the orientation of corners has either been estimated by finding the gradient direction of a corner patch [10], or rotating a corner template in the spatial domain for every possible angle combination [11]. However, these approaches are either not very accurate, or require significant computation power to be performed. The development of the design theory of multisteerable filters provided a significant increase in the possibility of efficiently determining the orientation of corner features in images. Once these corner features and their orientation angles are detected in images, a homography matrix can be determined by mapping one set of points in image A to another set of points in image B to transform the images into the same space [2], [12]. Since many computer vision techniques involve detecting many corners in images, and their respective orientation angles, the efficiency with which these corners can be detecting is extremely important.

Many Driver-Assistance (DA) systems also benefit from the research in feature detection using multisteerable features. With the advent of Google's driverless car, driver-assistance systems are becoming an important area of technological advancement. In particular, steerable filters can be used to determine the lane a driver is currently in, which is important for autonomously navigating the road [3]. Such approaches have been based on detecting edge features using a small subset of single steerable filters [3]. However, this approach relies on the assumption that roads are well marked, which is not always true in many cities. In many areas of the world, roads are imperfect; thus, more complicated features must be used to effectively and autonomously navigate the roads in real time. In this case, the computational complexity of detecting features becomes a constraint on the DA system since decisions must be made in real time, or fatal accidents could occur. For this reason, improvements in the efficiency with which multisteerable filters can be applied to detect features and their orientation in images could have a significant impact on DA systems.

Other important applications of steerable filters include exciting topics such as coding facial expressions [4], flow estimation in 3D images [5], and detecting cerebral vessels [6]. Using a small subset of low-level features and combining them can provide estimates of facial expressions people use when they are experiencing certain emotions such as happiness, sadness, or surprise [4]. Additionally, steerable filters can be used to estimate the flow between one image scene and another. This is similar to image stitching because features are detected in one image, and then found in a subsequent image. This approach can be used to track the movements of a target within an image, and could have many important applications such as surveillance, tracking of aircraft, or tracking movements of fish. Finally, steerable filters also have many important prospects in the medical field. Research has been done on uses of steerable filters to detect cerebral vessels in the human body using 3D medical imaging technologies [6].

These computer vision techniques cover a diverse area of applications, but they all can benefit from steerable filter implementations. Much research has been done on single steerable filters which are oriented in a single direction; however, until recently, theory of the design of multisteerable filters for multioriented features had not been done. Multisteerable filters can be used to efficiently detect the orientation of features in images, thus all of the applications listed in this section can benefit from improvements for the efficiency of detecting features. This thesis explores ways to improve the efficiency of detecting corners in images and efficiently determine the orientation angles that characterize these angles. Furthermore, optimization techniques are presented and compared. With improvements in the efficiency of applying multisteerable filters to images, many computer applications could benefit, and applications that were previously not possible due to computation complexity may become possible.

CHAPTER 2

THEORY OF MULTISTEERABLE MATCHED FILTERS

2.1 Basic Concepts and Definitions

Many computer vision problems involve the detection of *features*, which are typically patterns in images which may be oriented in multiple directions. Gathering useful information in images about the structure of a scene typically involves the detection of low-level features such as lines, edges, corners, and junctions. Applications involving feature detection are many, and include tasks such as target tracking, contour detection, homography transformations, image stitching, and camera calibration.

One common method for detecting low level features is to simply rotate a pattern template directly in the spatial domain for every possible angle orientation. Moreover, many features, such as corners, are defined by multiple orientation angles, which makes the spatial approach computationally intensive. Steerable filters can be used to significantly speed up the process of feature detection in images.

Steerable filters are filters which can be steered in any direction through some simple mathematical operations. The process by which a feature is synthesized at a specific orientation angle is referred to as *steering.*

In the past, steerable filter approaches were limited to a single orientation angle. However, many important image features are characterized by two or more orientation angles. A *multisteerable* filter is defined as a filter which can be efficiently steered in two or more directions. More recently, a new technique for efficiently implementing multisteerable matched filters at arbitrary rotation angles was presented for grayscale images [9]. The information presented in this chapter is an explanation of this research.

2.2 Background Information and Past Approaches

One solution for detecting low level features is to simply filter the image with a set of template kernels in the spatial domain, rotated at certain orientations. As previously mentioned, this approach is not only more computationally intensive, but many low-level features can appear at arbitrary rotation angles within an image. If a feature is at an orientation angle that is not one of the pre-computed kernels used, this feature detection method could fail.

For features which are simple and only characterized by a single orientation angle, such as edges, Jacob and Unser showed that filter designs based on steerable filters can efficiently detect low-level features [13], [14]. The rotated filters are then composed of a weighted sum of predetermined base filters [14]. The problem with this approach is that many important features in images, such as corners, are comprised of at least two orientation angles.

Much research has already been done in an effort to extend the concept of single steerable filters to multisteerable filters. These approaches are based on eigensystems, where the orientation is defined as a vanishing directional derivative. In these approaches, the minimum number of directional derivatives that are needed to make the observed signal vanish is correlated with the number of orientation angles of the feature [15], [16], [17], [18], [19], [20], [21], [22], [23], [24]. While these approaches can successfully detect two or more orientation angles that compose a feature, cannot distinguish between a corner, L-junction, T-junction, X-junction, or checkerboard feature [9]. One solution for detecting the orientations of multi-oriented features, while still being able to distinguish between different patterns oriented in the same directions, was proposed by Muhlich, Friedrich, and Aach, and is based on modeling these features [9].

2.3 Design of Single Steerable Filters

Steerable filters of a single orientation can be approximated by a template that is polar separable. These single-oriented filters can then be combined to produce a multisteerable filter, such that the relationship of the orientation angles of the individual single-oriented filters directly relates to the multiple orientations of the resultant multisteerable filter [9]. By inserting single steerable filters as arguments to a polynomial equation, and equating them to the desired pattern, multisteerable filters can then be determined by finding polynomial coefficients [9]. The resulting multisteerable filter is guaranteed to also be steerable [9]. This approach can be used to construct multi-oriented patterns like corners, L, T, and X-junctions. Thus, the problem of being able to detect the orientation angles of a pattern, but not the underlying pattern, is no longer present with this method [9].

2.3.1 Matched Filtering and Steerable Filters

The problem of detecting a template g in a feature f can be formulated as maximizing the correlation between the two patterns. In images, features often need to be detected at arbitrary rotation angles; therefore, the concept of matched filtering must be extended to rotated matched filtering [9]. To simplify notation, the rotation operator $(\cdot)^{\theta}$ is introduced to rotate a bivariate function by the angle θ as shown in equation 2.1.

$$g^{\theta}(r,\phi) = g(r,\phi-\theta) \tag{2.1}$$

This notation allows for the definition of rotated matched filtering. The problem of detecting a template g in an image f can thus be formalized in equation 2.2 by introducing A_{max}, which is a measure of how strongly the template is found in the image.

$$A_{max} = A\left(\vec{\theta}, x_0\right) = \max_{\theta}([f(x) * g^{\theta}(x)]_{x_0}) \tag{2.2}$$

with “*” denoting correlation, $[\cdot]_{x_0}$ meaning “evaluated at x_0”, and the template and image patches are both normalized to unit energy. The feature is said to be present at the point x_0 and in the direction $\vec{\theta}$ at points which exhibit sufficiently large local maxima. Steerable filters allow for an efficient method to compute equation 2.2.

2.3.2 Single Steerable Filters

Since many important low-level features have the property of polar separability, the template g can be split into two parts as shown below in equation 2.3.

$$g(r,\phi) = g_{dist}(r)g_{ang}(\phi) \tag{2.3}$$

One example of a polar separable template is an edge, which can be represented by the radial and angular function given below in equation 2.4.

$$g_{dist}(r) = \begin{cases} 1, & r \le r_{max} \\ 0, & r > r_{max} \end{cases}$$

$$g_{ang}(\phi) = \begin{cases} 1, & 0 \le \phi < \pi \\ -1, & -\pi \le \phi < 0 \end{cases} \tag{2.4}$$

The polar separability of the radial and angular function is further depicted in Figure 2.1.

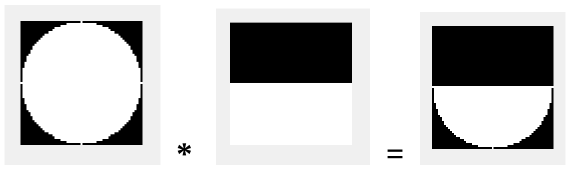

Figure 2.1: Polar Separability of a Single Steerable Filter

This is a single steerable filter. The filter can be steered by finding the Fourier coefficients of the single steerable filter, and multiplying them by $e^{-jp\theta}$, which rotates the feature by the angle θ. The Fourier coefficients of a polar separable filter can be found by using equation 2.5.

$$x_p = \sum_{-2P}^{2P} f\, e^{-jp\theta} \tag{2.5}$$

Where x_p is the pth coefficient of the steerable filter in the Fourier domain.

2.4 Design of Multisteerable Filters

The main idea for creating multisteerable filters is that two or more single steerable filters can be combined in such a way that they represent a multi-oriented template such that the resulting template is also steerable with two or more steering angles [9]. This chapter presents information that is mainly obtained from the work by Muhlich, Friedrich, and Aach [9].

2.4.1 Creating a Checkerboard Filter from Two Edge Filters

The checkerboard pattern is an important feature, since it is used in many computer vision applications such as camera calibration. This pattern is characterized by two orientation angles; thus, it can be detected using multisteerable filters. This section shows how the combination of two single steerable edge filters can result in a multisteerable checkerboard filter.

Let g_1 and g_2 represent two edge templates. The desire is to create a checkerboard filter k such that the checkerboard filter can be steered directly by the steering angles for g_1 and g_2. This approach can be expressed as shown in equation 2.6, where ○ is an unknown operator that needs to be found.

$$k^{\alpha,\beta} = g_1^{\alpha} \circ g_2^{\beta} \tag{2.6}$$

To find an operator that satisfies equation 2.6, Figure 2.2 is referenced. From this figure, an operator which satisfies equation 2.7 is desired. The only operator which satisfies this system of equations is the multiplication operator. Therefore, the multisteerable checkerboard pattern can be represented by the multiplication of two single steerable edge filters, and is also steerable by steering the edge filters individually.

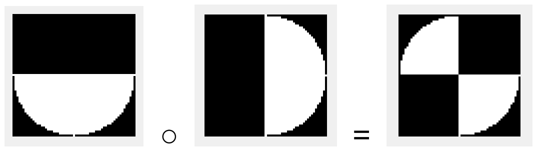

Figure 2.2: Creation of a Checkerboard Pattern from Two Edge Filters

$$-1 \circ -1 = 1 \qquad -1 \circ 1 = -1$$

$$1 \circ -1 = -1 \qquad 1 \circ 1 = 1 \tag{2.7}$$

Many low-level multisteerable features can be synthesized by simple multiplication. However, there are also many features which cannot, including corners. The method used to synthesize a multisteerable checkerboard feature must therefore be generalized further.

2.4.2 Properties of Multisteerable Filters

The previous section demonstrated that multi-oriented features must be represented as a product of single-oriented steerable filters. Additionally, single steerable filters can be summated to produce a multisteerable filter. Steerable filters are closed under multiplication and addition; therefore, the sum and product of two steerable filters is also steerable. Furthermore, a steerable filter is also steerable when multiplied by a scalar constant. Lastly, a constant mapping is steerable. These four properties of steerable filters are thus listed as follows:

- Property 1: g, h are steerable $\mapsto$ g·h is steerable
- Property 2: g, h are steerable $\mapsto$ g+h is steerable
- Property 3: g is steerable $\mapsto$ c·g is steerable
- Property 4: A constant mapping is steerable

These operations used in these four properties are the same operations used in a polynomial equation. Therefore, a multisteerable filter can be synthesized by solving a set of polynomial equations to find the desired pattern.

2.4.3 Synthesizing Multisteerable Filters from Polynomials

The previous section demonstrated how the properties of steerable filters hint at the idea of synthesizing multisteerable filters from polynomial equations. This section shows how this can be accomplished by using a corner template as an example.

Let p(g$_1$, g$_2$) represent a polynomial equation with two steerable filters as arguments. If using edge templates as the single steerable filters, then figure 2.3 shows the desired result.

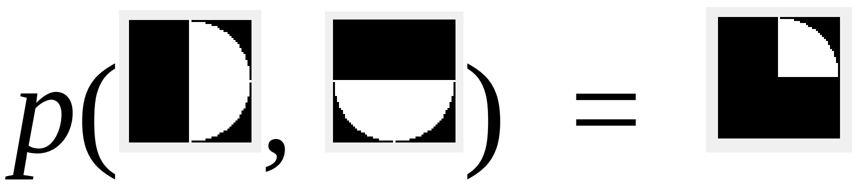

Figure 2.3: Synthesizing a Corner Template from Two Edge Templates

This equation can be modeled by a bivariate polynomial equation of degree 1 as shown in Equation 2.9.

$$p(x, y) = a_1 + a_2 x + a_3 y + a_4 xy \tag{2.9}$$

For any points x$_1$ in the first quadrant of g$_1$ and x$_2$ in the first quadrant of g$_2$, g$_1$(x$_1$) = 1 and g$_2$(x$_2$)= -1, which results in p=1. Similar, if a point is selected in each quadrant, and plugged into equation 2.9, the set of polynomial shown in Equation 3.10 is obtained.

$$p(1, -1) = \alpha_1 + \alpha_2 - \alpha_3 - \alpha_4 = 1$$

$$p(1,1) = \alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 = -1$$

$$p(-1,1) = \alpha_1 - \alpha_2 + \alpha_3 - \alpha_4 = -1$$

$$p(-1, -1) = \alpha_1 - \alpha_2 - \alpha_3 + \alpha_4 = -1 \tag{2.10}$$

Putting this set of polynomial equations into matrix form then yields the following system of equation shown in Equation 2.11.

$$\begin{bmatrix} 1 & 1 & -1 & -1 \\ 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \\ -1 \\ -1 \end{bmatrix} \tag{2.11}$$

Solving the system of equations shown in Equation 2.11 results in α_1 = -0.5, α_2 = 0.5, α_3 = -0.5, and α_4 = -0.5. Therefore, a corner can be represented by the following steerable filter shown in Equation 2.12.

$$k(r,\phi) = -0.5 + 0.5g_1 - 0.5g_2 - 0.5g_1g_2 \tag{2.12}$$

Moreover, equation 2.12 is composed of single steerable filters and constants which follow the properties of steerable filters explained previously. Therefore, the corner template in equation 2.12 is also a steerable filter. However, this polynomial method is based in the spatial domain, so it is no more efficient than prior approaches in the spatial domain. To design an efficient method for obtaining multisteerable filters, the design must be done in the Fourier domain.

2.4.4 Fourier Implementation of Steerable Filters

To implement a multisteerable filter in the Fourier domain, the Fourier coefficients of the individual single steerable filters must first be found. To do this, each base filter is simply multiplied by complex exponentials for each Fourier coefficient. This transform produces a Fourier coefficient vector of size 1 by 2p. Transforming these coefficients back to the spatial domain is done as shown below in Equation 2.13.

$$g^{\alpha}(r,\phi) = g_{dist}(r) \textstyle\sum_{P}^{P} x_p \, e^{-jp\alpha} e^{-jp\phi} \tag{2.13}$$

Next, if the product of two base filters is computed, the equation shown in Equation 2.14 is obtained.

$$g^{\alpha}(r,\phi) = g_{dist}^{2}(r) \textstyle\sum_{p=-P}^{P} \sum_{q=-P}^{P} a_p e^{-jp\alpha} \, b_q e^{-jp\beta} e^{-j(p+q)\phi} \tag{2.14}$$

Setting q = s – p, equation 2.14 becomes the equation shown below in equation 2.15. From this equation, it is evident that multiplying two image templates in the spatial domain is equivalent to convolving their 4P + 1 coefficients.

$$g^{\alpha}(r,\phi) = g_{dist}^{2}(r) \sum_{s=-2P}^{2P} \sum_{p=-P}^{P} (a_p e^{-jp\alpha}\, b_q e^{-jp\beta}) e^{-js\phi} \qquad (2.15)$$

Following the same method, it is known that adding two steerable filters is equivalent to adding their Fourier coefficients. Multiplying a steerable filter by a constant is the same as multiplying its Fourier coefficients by a constant. Lastly, adding a constant to a steerable filter is the same as adding the constant to the 0th Fourier coefficient, which is the DC offset term. Thus, all of the operations shown in section 2.4.2 can be implemented efficiently in the Fourier domain. Equation 2.12 in the spatial domain becomes Equation 2.16 in the Fourier domain, where a and b are the Fourier coefficients of two single steerable edge templates, and $\vec{o}$ is a vector of size 4P+1 whose center point is equal to -0.5 for the DC offset term.

$$k(r,\phi) = \vec{o} + g(\alpha) - 0.5g(\beta) - 0.5g(\alpha) * g(\beta) \qquad (2.16)$$

2.5 Additional Techniques

One of the problems with convolving Fourier coefficients is that the convolution operation turns two sets of 4P+1 coefficients into an 8P+1 coefficient vector. Therefore, nearly twice as many coefficients are obtained than what was started off with. However, if the innermost 4P+1 coefficients are taken to be result of the convolution, information from higher order Fourier coefficients is still included in the multisteerable filter. Furthermore, doing so produces a more accurate approximation of the multisteerable filter in the spatial domain than simply convolving 2P+1 Fourier coefficients [1].

However, truncating the Fourier coefficients in this way leads to the Gibbs phenomenon of undesirable oscillations [1]. To reduce such oscillations, the technique of windowing is applied to the Fourier coefficients. Windowing is a well-known technique from FIR filter design which produces smoother transitions and reduces oscillations. Many different kinds of window functions can be used, such as the Hamming window, the Hann window, the Bartlett window, and the Blackman window [1]. However, for this thesis, the author chose to use the Hamming window. The one disadvantage of windowing is that transitions from black to white are not as sharp. However, since the low pass characteristics of an imaging system prevent perfectly sharp transitions anyways, approximating the multisteerable filter in this way could actually improve performance on real images.

2.6 Similarity Function

Lastly, one problem with synthesizing multisteerable filters in this way is that the energy of each multisteerable filter is not constant due to interference effects between single steerable filter templates [1]. This could lead to problems with determining the optimal pattern match because one coefficient may have significantly more energy than the others, which biases this filter to be the best fit. The energy of a corner template can be expressed as a function of the difference of angles between the two single steerable filters. Therefore, the energy can be precomputed for every angle difference, and the multisteerable template can be normalized accordingly for more accurate detection.

In the spatial domain, feature detection can be expressed as maximizing the similarity, Q, function shown in equation 2.17, where $< f_1, f_2 >$ is the dot product of two images in the spatial domain.

$$Q(\alpha, \beta) = \frac{<I_{feature}, I_{template}>}{\sqrt{<I_{feature}, I_{template}>}\sqrt{<I_{template}, I_{feature}>}} \qquad (2.17)$$

The values of α and β which maximize the similarity function are the angles which best match the feature pattern. Therefore, the optimal corner template orientation is given by these two angles.

Similarly, a similarity function can be computed in the Fourier domain. Equation 2.18 shows the implementation of a similarity function in the Fourier domain. One thing to note is that this similarity function can be used for any filter as long as the Fourier coefficients can be found.

$$Q'(\alpha, \beta) = \frac{<\vec{v}_{feature}, \vec{v}_{template}>}{\sqrt{<\vec{v}_{feature}, \vec{v}_{template}>}} \tag{2.18}$$

Therefore, the similarity function can be computed directly from the Fourier domain. An advantage to this approach is that computing the similarity function in the Fourier domain is more efficient because only the dot product of coefficient vectors of size 4P+1 needs to be computed. In the spatial domain, the dot product of the entire feature and image patches needs to be computed, so the size of the coefficient vectors can be chosen such that optimizing the similarity function in the Fourier domain is always more efficient than doing so in the spatial domain.

To prove that the implementation in the Fourier domain from Equation 2.18 is more efficient that the implementation in the spatial domain in Figure 2.17, we can compute the number of multiplications and additions required for each method. The numerator of the similarity function for both equations contains most of the information needed for determining the optimal orientation angles, while the denominator is simply for normalization purposes. For this reason, I will analyze the number of multiplications and additions by simply considering the numerator.

If one considers the radius of the corner feature and image patch in the spatial domain for comparison purposes, the number of multiplications and additions required for the spatial domain implementation of the similarity function for a single orientation are $(2r+1)^2$ and $(2r+1)^2$-1 respectively. Thus, the number of multiplications and additions rises exponentially for the spatial domain implementation, and has a computational complexity on the order of $O(r^2)$.

Similarly, if one considers the 4P+1 coefficients for the Fourier domain implementation for comparison purposes, the number of multiplications and additions required for the Fourier domain implementation of the similarity function for a single orientation are 4p+1 and 4p respectively. Thus, the number of multiplications and additions rises linearly for the Fourier domain implementation, and has a computational complexity on the order of O(p).

Table 2.1 shows a comparison of the number of Multiplications and Additions for each implementation. From empirical evidence based on simulations of the Fourier domain implementation, it was determined that fewer than p=9 coefficients are required for very accurate results. However, the spatial domain implementation may require a significantly higher value for the radius, depending on the scale of the image.

Table 2.1: Comparison of Spatial Domain and Fourier Domain Implementations

Spatial Domain			Fourier Domain		
r	Multiplications	Additions	p	Multiplications	Additions
1	9	8	1	5	4
2	25	24	2	9	8
4	81	80	4	17	16
8	289	288	8	33	32
16	1089	1088	16	65	64
32	4225	4224	32	129	128
50	10201	10200	50	201	200

Setting the 4p+1 multiplications required in the Fourier domain equal to the $(2r+1)^2$ required in the Spatial domain equal when p=9 and solving for r shows that a radius of 2 or less is required for the spatial domain implementation to achieve similar computation complexity. Furthermore, a radius of 2 or less corresponds to an image size of 5 by 5 or less, which is typically not a large enough image patch to be considered a corner. Therefore, as long as the value of p is set to 9 or less, the Fourier domain implementation of multisteerable filters can be considered much more efficient than the spatial domain implementation. Additionally, the Fourier domain implementation allows for an efficient method of reducing large image features to a smaller size

CHAPTER 3

Efficient Implementation and Optimization

3.1 **Overview**

In this section, we will investigate efficient methods for implementing a corner detection algorithm using multisteerable filters. The research presented in this chapter is primarily from the work of the author of this thesis. Many image processing techniques involve detecting corners, and estimating their orientation. For example, stitching two images of the same scene from different viewpoints can be done by detecting corners in each image, estimating their orientation, and solving for the homography matrix that best transforms one set of corners into the other set of corners.

Historically, methods for estimating corner orientation have been slow because they were based mostly on brute force methods. For single image application, such slow speeds are acceptable, but often there are large databases of images for which corners must be detected and compared. In this case, there is a desire for a much faster corner pattern detector, and multisteerable filters can significantly speed up corner estimation through different optimization techniques.

3.2 **Modifications to Polynomial Method**

Section 2.4.3 shows how a corner template can be synthesized by solving a system of polynomial equations for the desired template. However, there is one issue with this approach. Namely, the corner templates for α=0°, β=90° and α=90°, β=90 will produce different corner templates as shown in Figure 3.1.

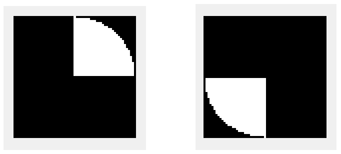
Figure 3.1: Examples for α=90°, β=0° (left) and α=0°, β=90 (right)

Similarly, the corner templates for α=180°, β=270° and α=270°, β=180 would also produce the same set of corner templates shown in Figure 3.1. This poses a problem because a template can have two sets of orientation angles which perfectly characterize the same template. To solve this problem simply setting the angle α to be equal to the larger of the two angles before synthesizing the corner template, as shown in Equation 3.1, will ensure that only the desired pattern is the one that is obtained.

$$(\alpha, \beta) = \begin{cases} (\alpha, \beta) & if\ \alpha \geq \beta \\ (\beta, \alpha) & if\ \beta > \alpha \end{cases} \tag{3.1}$$

3.3 **Storing Coefficients**

Another advantage of implementing an edge filter in the Fourier domain is that the single steerable edge templates can be stored directly in memory to be used as inputs to the multisteerable filter. This is done by pre-calculating the coefficients for all 360 possible rotations of an edge template, incrementing by the desired accuracy. For our simulations, we used an accuracy of 0.001°.

Furthermore, the last term in equation 2.16, which is the correlation of the two edge feature vectors, can also be pre-computed. This can be done by simply fixing one edge template at a 0° rotation, and incrementally increasing the other edge template by the desired accuracy until all options are covered. Then this correlation result can be steered to the correct orientation based on the values of α and β. However, some care needs to be taken to ensure the correct results.

If the difference angle between α and β in Figure 2.2 is greater than 180°, then the white and black areas are reversed. In this case, the angles α and β must be reassigned as shown in Figure 3.2 to ensure the correct multisteerable filter is generated. Then the coefficients should be steered to the magnitude of the difference angle of the new values of α and β.

$$(\alpha, \beta) = \begin{cases} (180^{\circ} + min(\alpha, \beta), max(\alpha, \beta) - 180^{\circ}) & if\ |a - b| \geq 180^{\circ} \\ (\alpha, \beta) & otherwise \end{cases} \quad (3.2)$$

Storing the pre-computed computed values for the arguments in Equation 2.16 in this way significantly speeds up the process of computing corner templates by around 3 times. However, further improvements might be achievable by exploiting the parallelism of Equations 2.15 and 2.16 by using the Graphics Processor Unit (GPU) to compute and apply corner templates in parallel. In this case, the GPU would most likely significantly outperform the method of storing coefficients due to the fact that the speed of accessing Random Access Memory (RAM) is slow, and often the bottleneck in speed for many applications.

3.4 **Corner Detection**

To determine where corners are in an image, first corner detection must be applied to the image. The corner detection method used in this thesis is the Harris corner detection algorithm. Typically, a Gaussian window is first applied to smooth the image, which makes corners stand out more; however, in this case a Gaussian window made the exact point of the corner off by a significant amount. If the detected point of the corner is not sufficiently close to the real corner, the similarity function will not give accurate results in orientation estimation.

To determine where corners are located in an image, the Harris corner detection algorithm shown in Equation 3.3 was used. The constant k can be any real value between 0.04 and 0.06, but for this thesis the value 0.24 was used. The corners which gave the top 50 responses were then selected as corners.

$$R = Determinant - k * Trace \tag{3.3}$$

Another Harris corner detection algorithm is to use the determinant divided by the trace as a measure of corners. However, in this case, the trace can be 0 which will result in the response being infinite. To avoid such issues, the corner detection shown in Equation 3.3 was used.

3.5 **Optimization Methods**

The similarity function from Equation 2.18 can be solved for the orientation angles which give the highest response by using different optimization methods. In this thesis, I tried three different optimization methods, and compared their results. To compare the results of different optimization methods, I used a brute force method to compute the global optimal, and used these results as the ground truth. The optimization methods that were tested include gradient descent optimization, particle swarm optimization, and Levenberg-Marquardt optimization. The Levenberg-Marquardt algorithm was implemented, and subsequently it was found to be an ineffective method for this similarity function. This section explains the theory of these different optimization methods, as well as specific information on the implementation in this thesis.

3.5.1 **Gradient Descent Optimization**

Gradient descent optimization is an optimization method for finding the minimum of a multivariate function by travelling in the gradient direction. As long as the gradient is a non-zero vector, the gradient vector will be orthogonal to the tangent vector to a curve going through x_0 on the level set f(x) = c [14]. Therefore, the direction of maximum increase is in the gradient direction, so for minimization problems the minimum is in the negative gradient direction. Equation 3.4 below shows the equation for updating the coordinate x for a function f(x) by subtracting the gradient. The argument α can be set to some value, or dynamically set to decrease depending on the number of iterations. The same principle applies to multivariate functions such as the similarity function in Equation 2.18. For maximization problems, the objective function can simply be inverted to solve for the minimum.

$$x^{k+1} = x^k - \alpha_k \nabla f(x^k) \tag{3.4}$$

The gradient descent method stops updating the value of x once the value of the gradient is 0, at which point a local minimum is found. However, in practicality, the stopping criteria for updating x can be if the gradient is below some small value. For this thesis, if the value of the gradient dropped below the accuracy of the pre-computed single steerable filters, the gradient descent method stopped. This means that if the gradient dropped below 0.001, the gradient method would stop.

For the implementation in this thesis, the value of α was initialized to a value of 16, and decremented by 1 until it reached a value of 1. Care must be taken to not decrease α too much or else the Gradient method may never converge. The value of a was initialized to a larger value at the start of the gradient descent method in order to allow the x to incremented in larger steps when it is far away from the solution. Furthermore, the value of x is randomly initialized between (0°,0°) and (360°,360°), so the current x could be far away from the optimal solution. After a few iterations of the gradient descent method, x should be much closer to the optimum, so a can be set to 1 so as to not overshoot the correct solution.

One issue with the gradient descent method is that if a function has local optimum, the gradient method can converge to local optima instead of the global optimum. This poses a problem in estimating the orientation of a corner. Figure 3.2 below shows one example of a cost function with multiple local maxima. In this example, a perfectly synthesized corner with steering angles α=269.1013°, β=328.4598° were used as the feature pattern. Then, a brute force analysis of the objective function at every rotation angle was obtained. The areas that are more red in color correspond to areas with greater values of the cost function, while areas that are more blue correspond lower values.

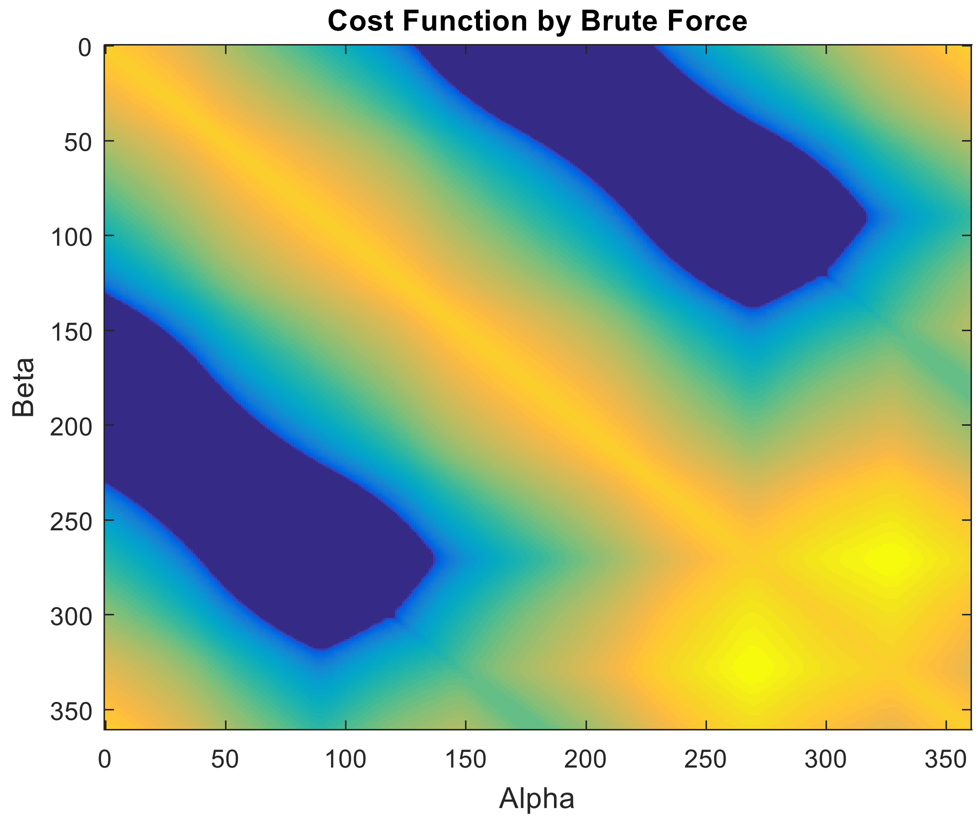

Figure 3.2: Objective Function with Multiple Local Maxima

From this graph, it is evident that the objective function has multiple local maxima. The global maxima are correctly identified near the correct global maximizer. However, there is also a significant local maximum around α=0°, β=360° and α=360°, β=0°. Furthermore, there are many local maxima whenever the difference between α and β is around 0°. The gradient descent method has trouble converging to the global optimum in these cases unless it is randomly initialized close to the global optimum.

To explain this phenomenon, we must take a look at the templates generated from these orientation angles. The feature oriented at α=269.1013°, β=328.4598°, and an image template oriented at α=0°, β=360° are shown in Figure 3.3. From this Figure, it is evident that the local optima are present when the template image is completely black. Corners with extremely small differences in angles α and β will similarly produce a black template.

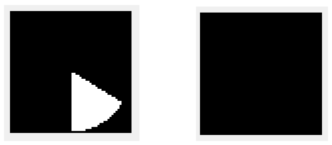

Figure 3.3: Comparison of Feature with Template

Next, we should take a look at how the template looks when difference of the orientation angle is decremented or increment. Figure 3.4 shows two templates where the value of α is incremented and decremented by 30°. From this Figure, we can now see why a completely black template yields a local optimum in the similarity function. The similarity function is a measure of how similar one feature vector is to another, and clearly a black template is more similar to the feature than the surrounding points. This is because similarity is roughly defined as the number of pixels which are the same value between images.

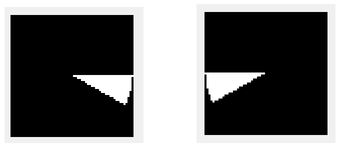

Figure 3.4: Decrementing α by 30° (left) and Incrementing α by 30° (right)

One simple method of solving this problem is to randomly reinitialize α and β when the angle difference is extremely small, or close to 360° apart. In this way, we can attempt to force the solution to converge near the global optimum. Since these angle pairs correspond to black image templates anyways, the template is not a corner, and therefore has no corner orientation. In these cases, it should be obvious that the gradient descent method did not converge to the global maximum if the known image feature is known to be a corner.

However, there are also some situations where other local maximum can be present. In these cases, the second local maximum is typically close in value to the true global maximum, but significantly far away to pose problems with orientation estimation. Figure 3.5 shows one such case.

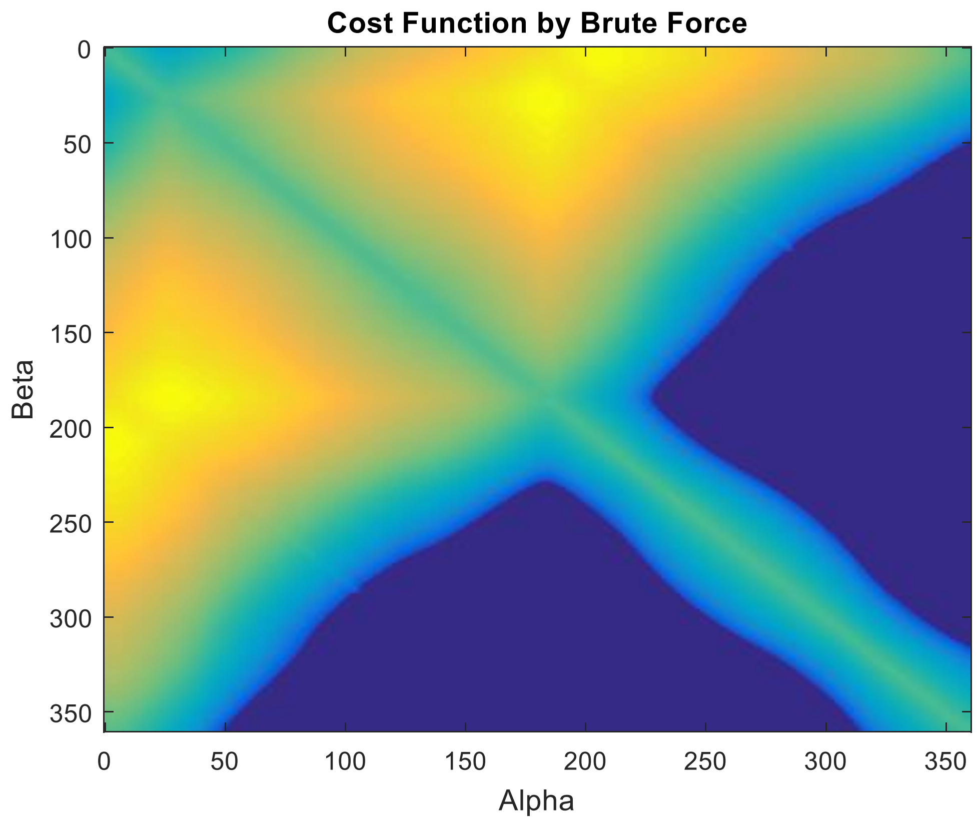

Figure 3.5: Objective Function with Indistinguishable Local Optimum

In this example, the true values of α and β are 210.0410° and 7.8983° respectively. One local maximum correctly is detected at the global maximum; however, a second local maximum is apparent around α=28°, β=185°. While this local maximum is close to the true orientation of the corner, this is a significant margin of error. This is one major problem with the gradient descent method.

In this thesis, instead of computing the gradient directly, an approximation of the gradient was obtained by computing the difference in the value of the similarity function at a point very close to the current point. This difference is a good approximation for the gradient when a function for the gradient is not determined. To approximate the gradient, the value of the similarity function is computed for α being incremented by a very small value. The same method is used to find the gradient in the β direction.

3.5.2 Particle Swarm Optimization

Partical swarm optimization is an optimization method that was introduced by a sociologist and engineer working together [14]. The partical swarm algorithm was created as an attempt to mimic the social behavior of animals such as bees, birds, and wildebeest [14]. Instead of updating a single candidate solution, as in the case of the gradient descent method, partical swarm optimization involves updating a set of candidate solutions all at once, called a *swarm*.

The basic idea of partical swarm optimization is that some candidate solutions in the set will travel toward the global best solution, others will travel towards their local best, and others will travel in the same direction they were previously travelling in. This idea based on social patterns can be modeled as shown in Equation 3.5.

$$x_{p+1} = r_1 c_1 V_p + r_2 c_2 (x_L - x_p) + r_3 c_3 (x_G - x_p) \tag{3.5}$$

In this equation, r_1, r_2, and r_3 are random numbers in the interval 0 ~ 1. Similarly, c_1, c_2, and c_3 are constants in the same interval. Finally, x_p is a candidate solution at the pth iteration, x_L is the local best solution for this partical, and x_G is the global best solution for all particles. Finally, V_p is the velocity of this candidate particle at the pth iteration. To obtain the initial velocity, two random sets of particle swarms are generated, and the velocity is defined as the difference between these particles. For this thesis, the value of r_1 was set to always be equal to 1, and all of the constants were also set to be equal to 1. Thus, equation 3.5 simplifies into Equation 3.6.

$$x_{p+1} = V_p + r_1\left(x_L - x_p\right) + r_2(x_G - x_p) \tag{3.6}$$

For this implementation, I initially started off with 50 particles and 50 iterations. The solution which yields the largest value for the similarity function in Equation 2.18 is then taken to be the orientation of the corner. However, to improve the efficiency of this method, whenever an iteration of the partical swarm method did not result in a new global best, the particle which corresponded to the lowest value in the similarity function was dropped from the set of particle solutions.

One advantage the particle swarm method has over the gradient descent method is in parallelization. Since the particle swarm method tests many different values for α and β at once, all of these multisteerable filters as well as their corresponding values in the similarity function can be computed in parallel. The gradient descent method cannot be done in parallel because the gradient must be computed before the next candidate point can be determined.

CHAPTER 4

SIMULATION AND RESULTS

4.1 Synthesized Corners

First, perfectly synthesized corners were generated to compare optimization methods under ideal conditions. To test the two methods, they were compared with the brute force method being considered the ground truth. One thing to consider when comparing methods is that twice as many steering operations are needed for the gradient descent method since the steerable filter must be steered for α and β being incremented by a small amount in order to obtain an estimation of the Gradient. For this reason, a single iteration is defined to be a steering operation that is performed for each method. This is the fairest way to compare the two methods directly. For example, for the particle swarm method with 50 particles and 50 runs, 2500 iterations will be performed. Similarly, for the gradient descent method which converges after calculating the gradient 1250 times, 2500 iterations will be performed.

Table 4.1 shows a comparison of the number of iterations required for both the gradient descent and particle swarm optimization methods. For this comparison, 100 different simulations were run in order to get good idea of which method was more efficient. Most notably, the average number of iterations for the gradient descent method was over 4 times greater than the average number of iterations for the particle swarm method. Furthermore, the particle swarm method was fairly consistent since in the worst case it only needed 23 more iterations than the average number of iterations to converge. However, the gradient descent method was wildly inconsistent, requiring over 5 times the average number of iterations in the worst case.

Table 4.1: Comparison with Ideal Conditions for 100 Trials per Case

		Average	Worst	Local	Global
	Trials	Iterations	Case	Max	Max
Gradient Descent	100	1688.74	8720	13	87
Particle Swarm	100	419.43	443	0	100

Another thing to note is that the gradient descent method actually converged to a local maximum which was near the global maximum, but significantly far away from the global maximum, 42 times out of 100. Moreover, the particle swarm method converged to a point very close to the global maximum in every case. When the gradient descent method did converge to the global maximum, however, it was closer to the results obtained from the brute force method than the particle swarm method. This result is to be expected since the particle swarm method is based on randomization. However, such accuracy is not possible with this implementation since even the global optimum obtained from the brute force method is typically within 2° of error. To explain why the particle swarm method is clearly more efficient than the gradient descent method, two examples are presented in the following sections

4.1.1 Example 1

In this example, the values of α and β were randomly chosen to be 184.6626° and 138.3290° respectively. Figure 4.1 shows the generated corner, as well as the detected corner orientations for each method. From this Figure, it is evident that both methods converged to the correct solution. Table 4.2 provides a more in depth analysis of the accuracy of the results for each method.

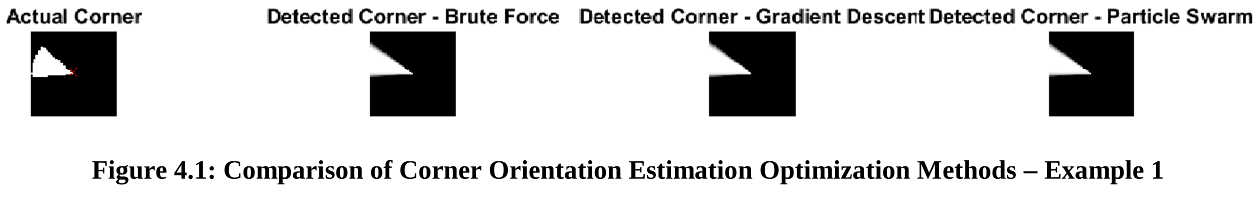
Actual Corner
Detected Corner - Brute Force
Detected Corner - Gradient Descent
Detected Corner - Particle Swarm

Figure 4.1: Comparison of Corner Orientation Estimation Optimization Methods – Example 1

Table 4.2: Accuracy Comparison of Optimization Methods

	Actual Corner	Brute Force	Gradient Descent	Particle Swarm
Alpha	184.6626	184.422	184.04	183.267
Error (α)	-	0.2406	0.6226	1.3956
Beta	138.329	138.018	140.403	140.834
Error (β)	-	0.311	2.074	2.505

From Table 4.2, it is evident that the gradient descent method is more accurate than the particle swarm method for this example. However, both optimization methods are 2° away from the actual corner in the β orientation. Furthermore, neither method converged to a solution as close to the actual corner as the brute force method, which is expected. However, the accuracy of these methods is most likely suitable for most applications of corner orientation estimation.

To compare the efficiency of both optimization methods for this example, a graph of the number of iterations versus the distance from the global best of the similarity function as determined by the brute force method is shown in Figure 4.2. In this case, it is clear why the gradient descent method is less efficient than the partical swarm method. The particle swarm method converged in only 402 iterations, while the gradient descent method took 1378 iterations to converge. The main reason that the gradient descent method is slower than the particle swarm method for this example is that the gradient descent method had to reinitialize 4 times for reasons explained in section 3.5.1. However, it is clear from this graph that the gradient descent method converges at a much slower rate than the particle swarm method.

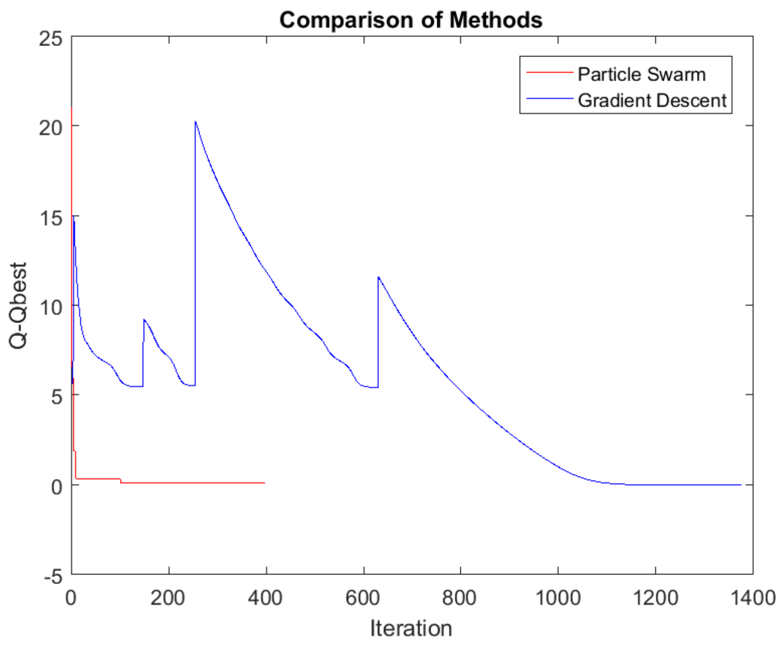

Figure 4.2: Comparison of Iterations Required Between Methods – Example 1

4.1.2 Example 2

In this example, the values of α and β were randomly chosen to be 275.4343° and 136.1875° respectively. Figure 4.3 shows the generated corner, as well as the detected corner orientations for each method. From this Figure, it is evident that both methods converged near the correct solution. Table 4.3 provides a more in depth analysis of the accuracy of the results for each method.

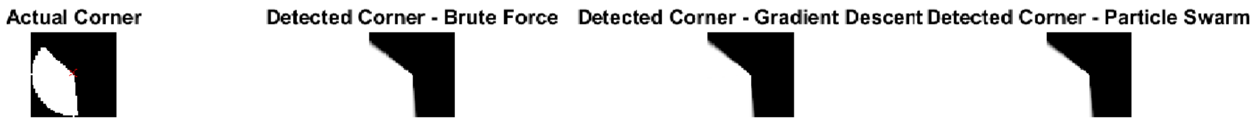
Actual Corner
Detected Corner - Brute Force
Detected Corner - Gradient Descent
Detected Corner - Particle Swarm

Figure 4.3: Graphical Comparison of Corner Orientation Estimation Optimization Methods – Example 2

Table 4.3: Gradient Descent Converges to Local Maximum

	Actual Corner	Brute Force	Gradient Descent	Particle Swarm
Alpha	275.4343	275.72	318.178	274.831
Error (α)	-	0.2857	42.7437	0.6033
Beta	136.1875	134.56	95.383	137.44
Error (β)	-	1.6275	40.8045	1.2525

From Table 4.3, it is evident that the gradient descent method converged to a local maximum. However, this angle combination produced a corner template that closely matches the actual corner in appearance. This is because of the effect caused by the angle difference between α and β being greater than 180° mentioned in Section 3.2. Realistically, this is the same corner as the others because if one subtracts 180° from α and adds 180° to β for the gradient descent method, the corresponding errors are reduced to 0.0513 and 1.9905 respectively. Therefore, the accuracy for each optimization method is adequate in this example.

Figure 4.4 shows a comparison of the number of iterations required for each method in this example. In this case, the gradient descent method converged at a much faster rate than normal, and even converged faster than the particle swarm method overall. For this example, the gradient descent method converged in 398 iterations while the particle swarm method converged in 417 iterations.

However, Table 4.1 shows that these results are the exception. On average, the particle swarm method outperforms the gradient descent method in terms of efficiency by 4 times. This is mainly due to the fact that the gradient descent method has a large variance in the number of iterations required to converge, while the particle swarm method has a very small variance. These results can be explained by the fact that the gradient descent method must be initialized to a random number to begin, which can often be significant far away to the global maximum, or in the worst case close to a local maximum that is not the global maximizer.

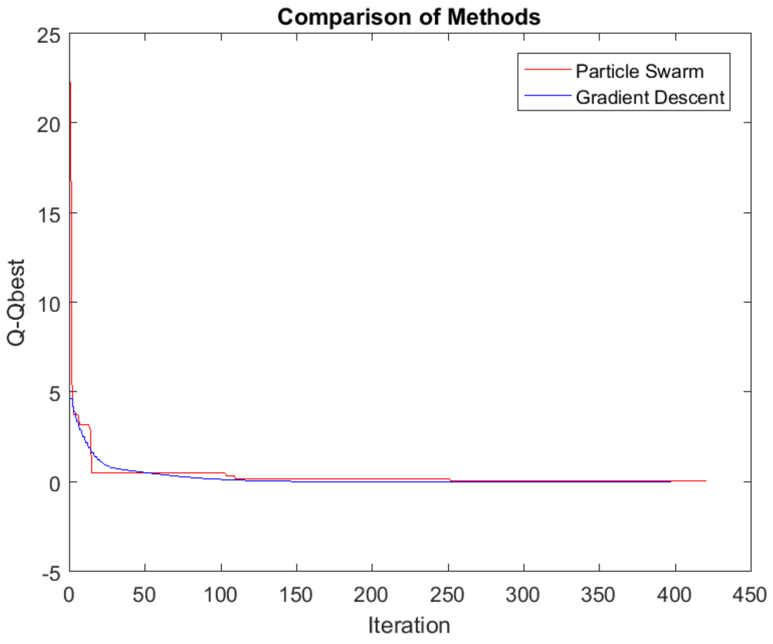

Figure 4.4: Comparison of Iterations Required Between Methods – Example 2

4.2 **Synthesized Corners with Added Noise**

Next, it was desired to test each optimization method under noisy conditions. Since imaging systems are not perfect, it is imperative that a corner orientation estimator must be robust to noisy conditions. Again, the two methods were compared with the brute force method being considered the ground truth. Four different types of noise were tested, which include Gaussian noise, Poisson noise, Salt and Pepper noise, and Speckle noise. The Gaussian noise had a mean of 0 and variance of 0.1, and the Salt & Pepper noise had a 5% noise density.

Table 4.3 shows a comparison of the number of iterations required for both the gradient descent and particle swarm optimization methods under these various noise conditions. For this comparison, again 100 different simulations were run in order to get good idea of which method was more efficient and more robust to different noisy conditions. Again, the average number of iterations for the gradient descent method was much larger than the average number of iterations required for the particle swarm method. Moreover, the gradient descent method seems much more prone to converging to a local maximum when noise is present, especially for Poisson, Salt & Pepper, and Speckle type noise.

Table 4.4: Comparison with Noisy Conditions for 100 Trials per Case

Noise	Optimization	Average Iterations	Worst Case	Local Max	Global Max
Gaussian	Gradient Descent	2008.5	5272	14	86
	Particle Swarm	420.16	451	0	100
Poisson	Gradient Descent	1846.56	6684	27	73
	Particle Swarm	407.29	432	0	100
Salt & Pepper	Gradient Descent	3386.28	10000	26	74
	Particle Swarm	418.54	448	0	100
Speckle	Gradient Descent	1909.4	5928	58	42
	Particle Swarm	419.68	463	7	93

For Gaussian noise added a synthesized perfect corner, there was no noticeable change in the accuracy of either method. The gradient descent method did take slightly longer to converge on average; however, the worst case converged significantly faster than when no noise was present. No significant change was seen with the particle swarm method when noise was added compared to a perfect corner.

When noise with a Poisson distribution is added to the corner feature, a significant increase in the number of times the gradient descent method converged to a local maximum over the global maximum occurred. The gradient descent method did converge on average faster than previously, but this may be due to the fact that there are more local maxima in the similarity function when this type of noise is present. Again, the particle swarm method showed no significant change when compared to the results obtained from using a perfect corner.

Noise of types Salt and Pepper and Speckle were where the biggest decrease in performance was noted for the gradient descent method. When Speckle noise was added to the corner feature, the gradient descent method converged to a local maximum over half of the time.

While the gradient descent method did show a significant statistical increase in the average number of iterations required to converge when Salt & Pepper noise was added, this method did not show a significant increase when Speckle noise was added. One thing to note about the gradient descent method is that it is set to stop running after 10,000 iterations have occurred to prevent the program from running indefinitely. Some of the trial runs for the Salt & Pepper method failed to converge based on the stopping criteria before 10,000 iterations were performed, which significantly impacted the average convergence rate for this method.

For the case of Salt and Pepper noise, the particle swarm method again showed no significant change from the perfect conditions. However, when Speckle noise was added to the corner, the particle swarm method did converge to a local maximum 7% of the time. This may be due to the fact that the area around the global maximum with values close to the global maximum is smaller under these types of noise conditions. Since this method is based on randomization, the probability of finding a point near the global maximum when the area surrounding the global maximum is smaller goes down. However, the particle swarm method should be more robust to noise if more iterations are performed.

Clearly, the particle swarm method is more robust to noise in the presence of all types of noise that were tested for in these trials. This may be due to the fact that there are more local maxima in the similarity function when noise is present, which could cause the gradient descent to converge to local maxima more often. Since the particle swarm method is based on randomization, however, it is less prone to converging to a local maximum. Since in the event Speckle noise is present in an image, both methods performed the worst, an example will be shown in the following section for this type of noise in order to get a better idea of why both methods are converging to local maxima more often.

4.2.1 Example 1 – Speckle Noise

In this example, the values of α and β were randomly chosen to be 221.1638° and 7.2180° respectively. Figure 4.5 shows the generated corner with speckle noise added, as well as the detected corner orientations for each method. From this Figure, it is evident that the particle swarm method converged near the correct solution, but the gradient descent method did not. Table 4.3 provides a more in depth analysis of the accuracy of the results for each method. In this example, the particle swarm method converged in 463 iterations, while the gradient descent method converged in 3478 iterations. While the particle swarm method still shows its advantage of being able to converge faster, in this example it also shows that it has a greater tendency to converge to a global maximum, especially when noise is present.

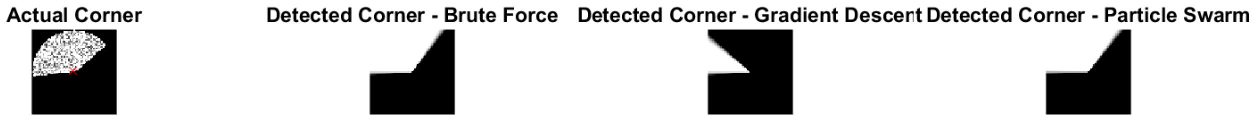
Actual Corner
Detected Corner - Brute Force
Detected Corner - Gradient Descent
Detected Corner - Particle Swarm

Figure 4.5: Graphical Comparison of Optimization Methods with Speckle Noise Present

Table 4.5: Accuracy Comparison Under Noisy Conditions

	Actual Corner	Brute Force	Gradient Descent	Particle Swarm
Alpha	221.1638	230.616	136.174	226.971
Error (α)	-	9.4522	84.9898	5.8072
Beta	7.218	3.794	183.178	4.429
Error (β)	-	3.424	175.96	2.789

From Table 4.5, it is evident that the gradient descent method converged to a true local maximum. While the gradient descent method converged an orientation that is oriented in a similar direction as the actual corner, the overall error is quite large because it did not converge to the correct point. A graph of the similarity function for this example is shown in Figure 4.6.

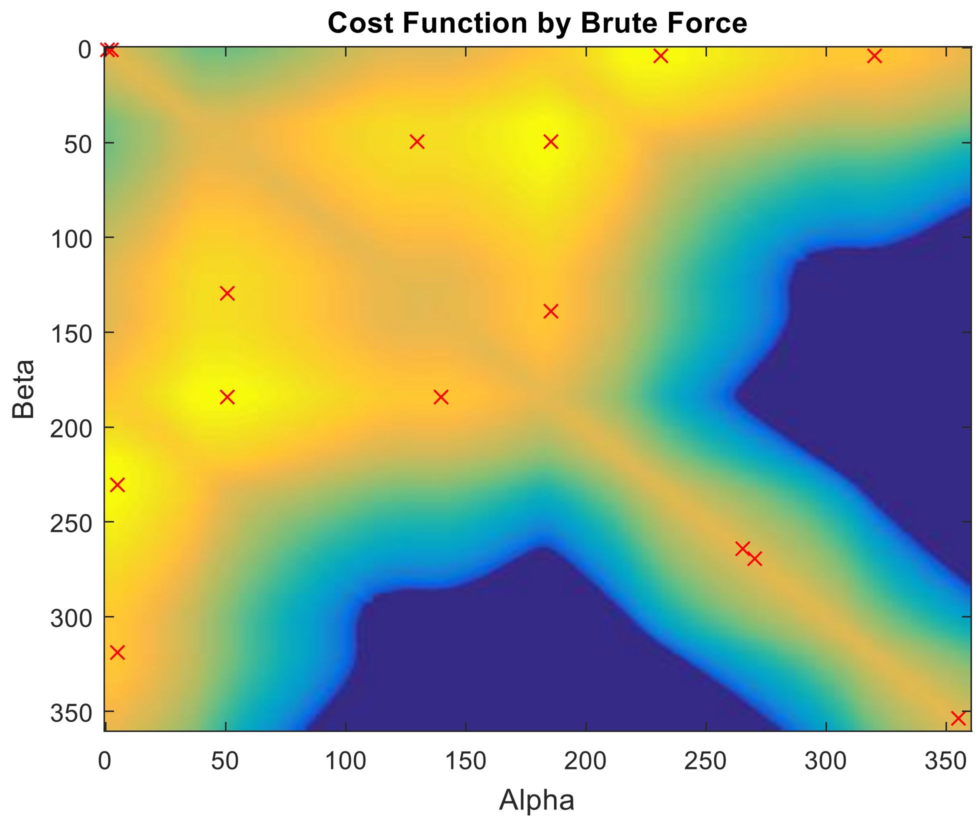

Figure 4.6: Similarity Function of Corner with Speckle Noise

Figure 4.6 shows the similarity function obtained from the brute force method. The local maxima were obtained, and plotted as red crosses on the figure. From the figure, we can see that there are many local maxima in this image. Figure 4.7 shows the graph in Figure 4.6 zoomed in around the point where the gradient descent method converged. While clearly there are larger maxima oriented in the -α direction, the point on which the gradient descent method converged is a region in the similarity function that is relatively flat. For this reason, the gradient will never be large enough to cause a move toward convergence in the direction of the global maximum.

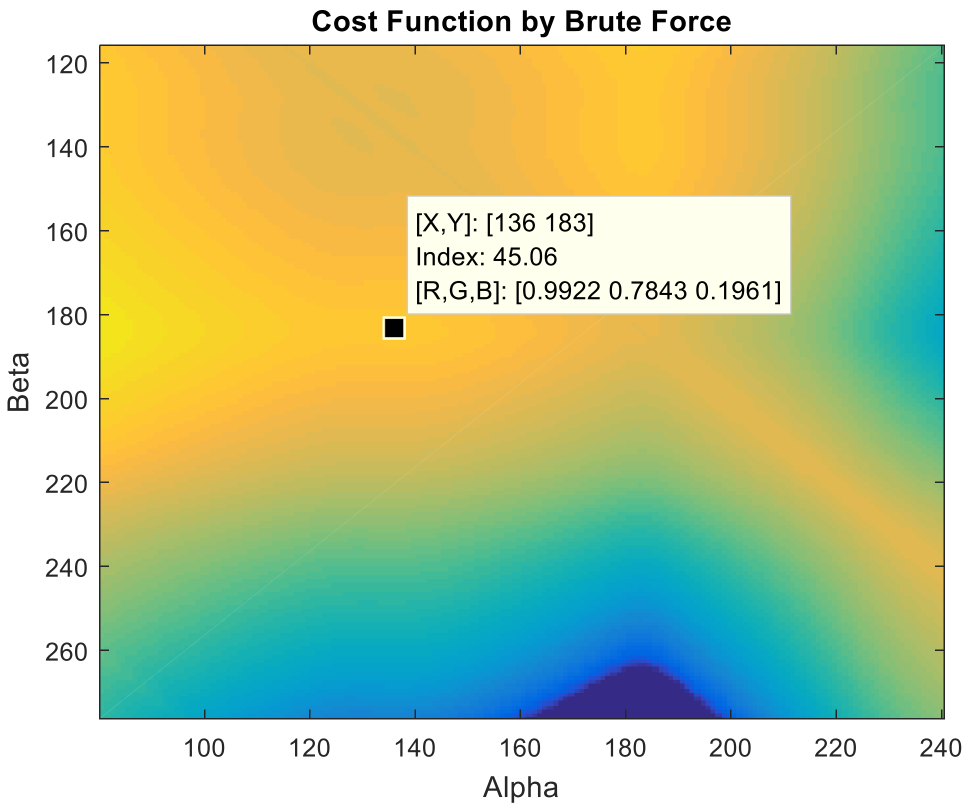

Figure 4.7: Similarity Function Zoomed in Around Convergence Point

For comparison, Figure 4.8 shows the cost function for an ideal corner with no noise for the same values of α and β. Again, the local maxima are plotted on top of the similarity function as red crosses. For the ideal corner, there are still local maxima, but they appear at small angle differences, which would cause the gradient descent method to reinitialize, but not converge at those local maxima. Thus, it is apparently that noisy conditions may cause extraneous local maxima to appear in the similar function, which may cause problems for the convergence of the gradient descent method to the global maximum. Real images are captured from imaging systems that are not perfect, and thus there is noise present in these images. For this reason, an optimization method based on randomization such as particle swarm is expected to perform much better on real images.

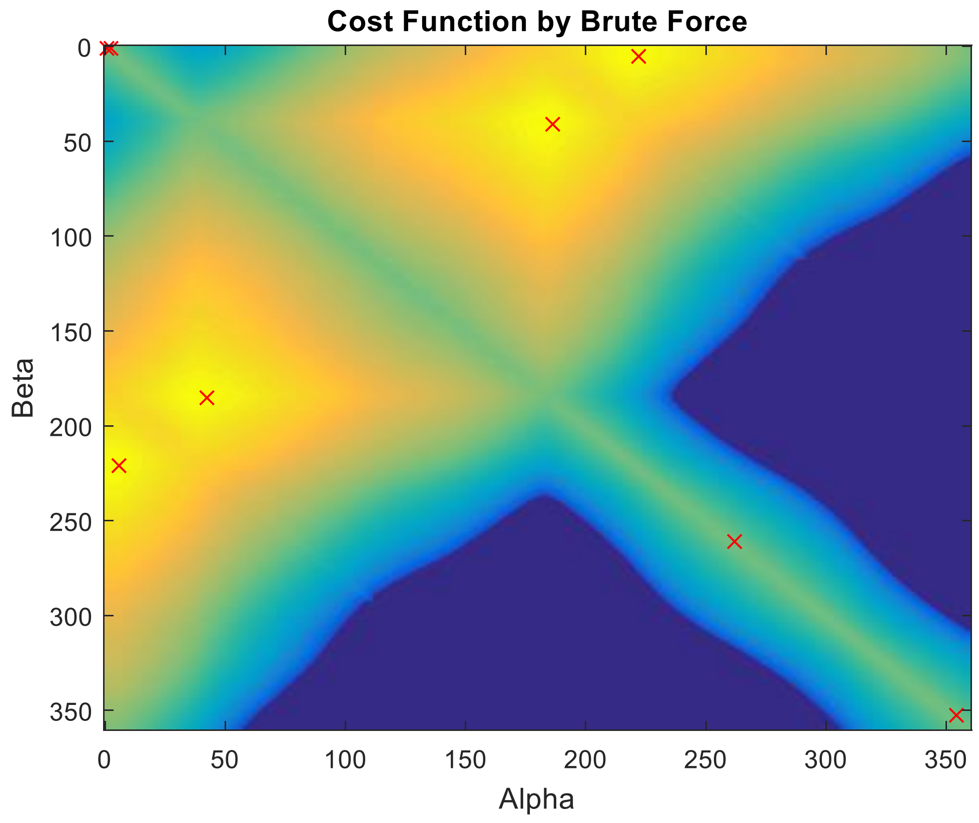

Figure 4.8: Similarity Function of Corner with No Noise Added

4.3 **Corners in Real Images**

Next, it was desired to test each optimization method using corners from real images. Figure 4.9 shows 6 different images that were used to test these optimization methods. Some of the images contain simple patterns where the corners should be easily apparently, while others are more complicated images from the real world. In all of these cases, the images were first converted to grayscale images since this method only works on grayscale images. Then the Harris corner detection algorithm was used to obtain the coordinates of corners in real images. A window of size 61 by 61 around the corners was used as input to the orientation detection algorithm.

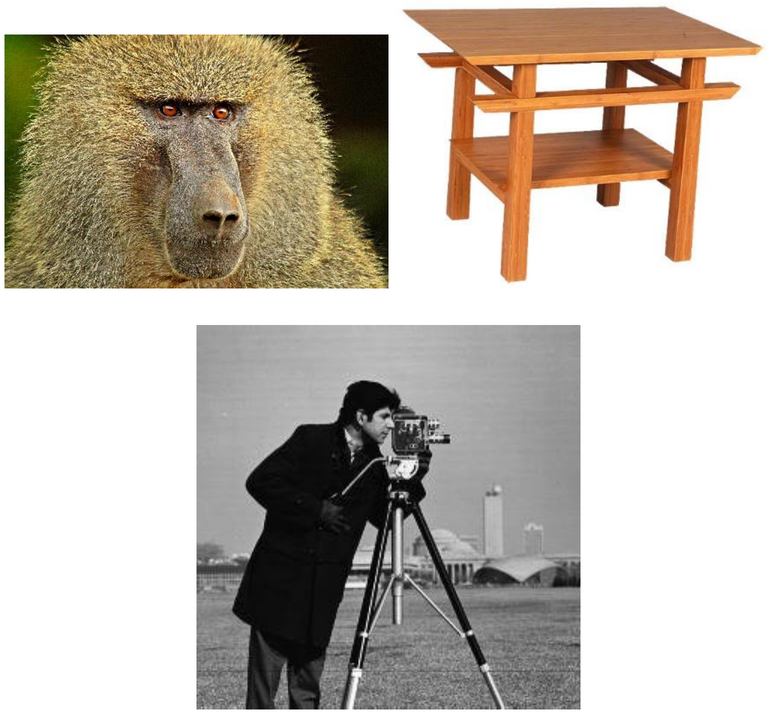

Figure 4.9: Images Used for Real Corners

4.3.1 Example 1 - Corners in Baboon Image

In this example, corners were found in the image of the baboon. Figure 4.10 shows the corner in the original image as detected by Harris corner detection, as well as the detected corner orientations for each method. From this figure, it is evident that all of the optimization methods converged to the same solution on this corner from a real image. We can also see from this Figure that the corner that was detected is quite different from the types of corners that were synthesized. In this case, the orientation of the corner could just have been determined to be shifted 180° from what was detected. This problem is caused because of what we consider to be a corner. In reality, every pixel in an image could be considered to be a corner, but tradeoffs must be made in order to use corners as features in images. Figure 4.11 shows a comparison of the number of iterations required for each method. Again, particle swarm converges much faster than the gradient descent method.

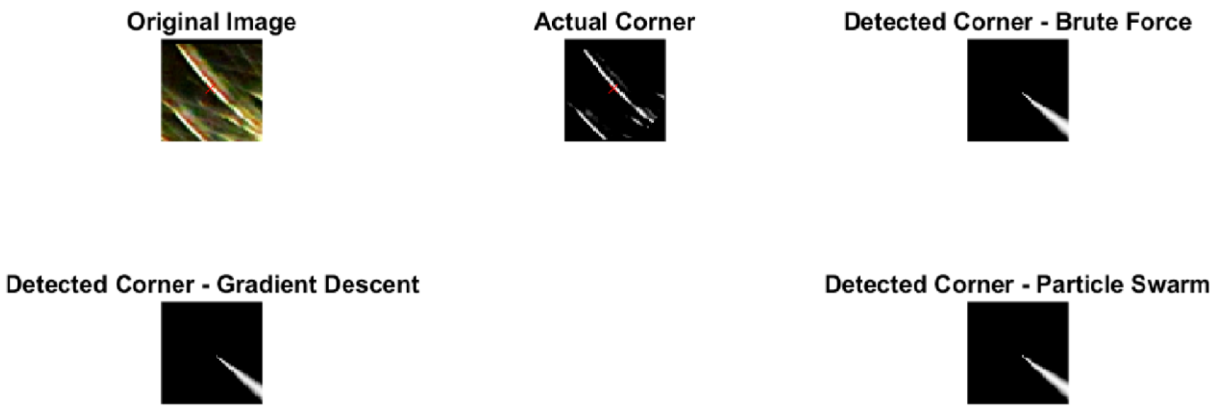
Original Image
Actual Corner
Detected Corner - Brute Force
Detected Corner - Gradient Descent
Detected Corner - Particle Swarm

Figure 4.10: Graphical Comparison of Optimization Methods on Baboon Image

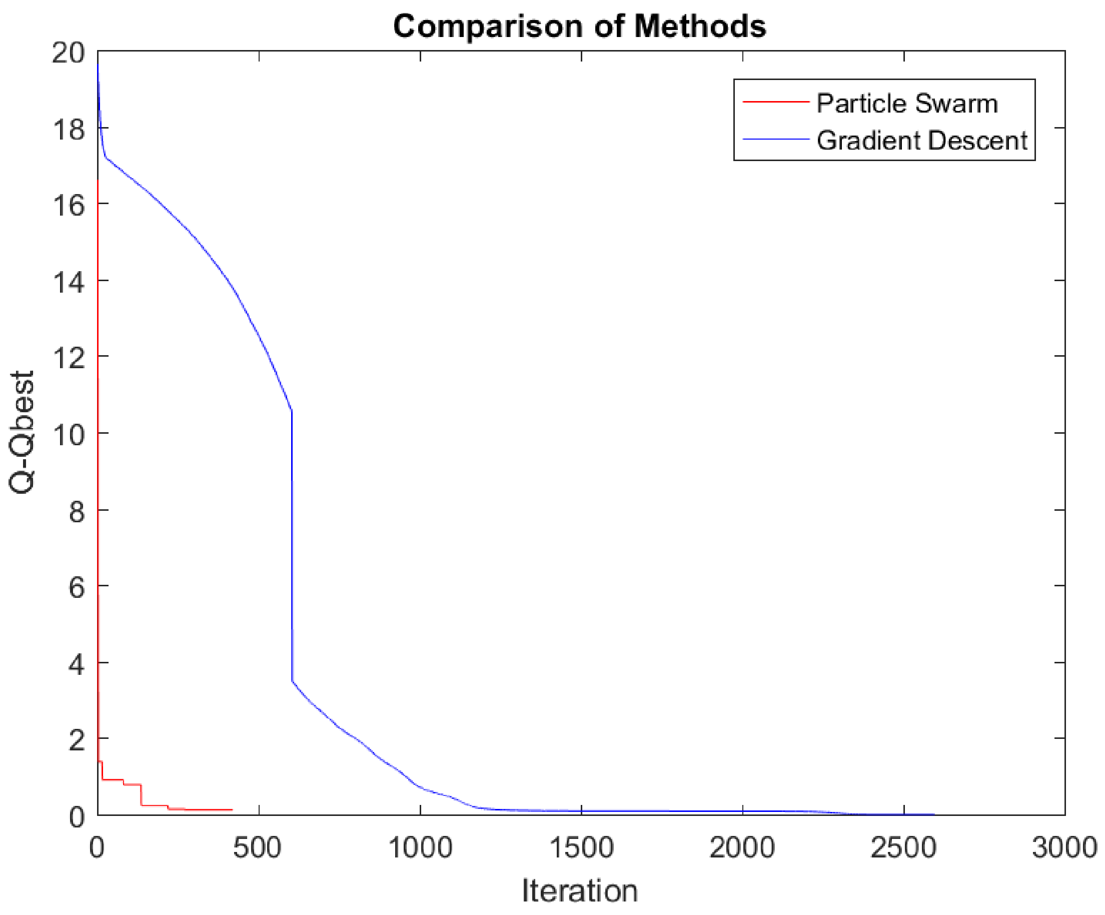

Figure 4.11: Comparison of Iterations Required Between Methods

4.3.2 Example 2 - Corners in Stool Image

In this example, corners were found in the image of the stool. Figure 4.12 shows the corner in the original image as detected by Harris corner detection, as well as the detected corner orientations for each method. From this figure, it is evident that all of the optimization methods converged to the same solution on this corner from a real image. The detected corners, however, closely resemble the corner in the original image. We can see that the left edge of the corner is slightly skewed to the left as opposed to the original image, but this is most likely caused by the fact that the center of the corner is slightly off.

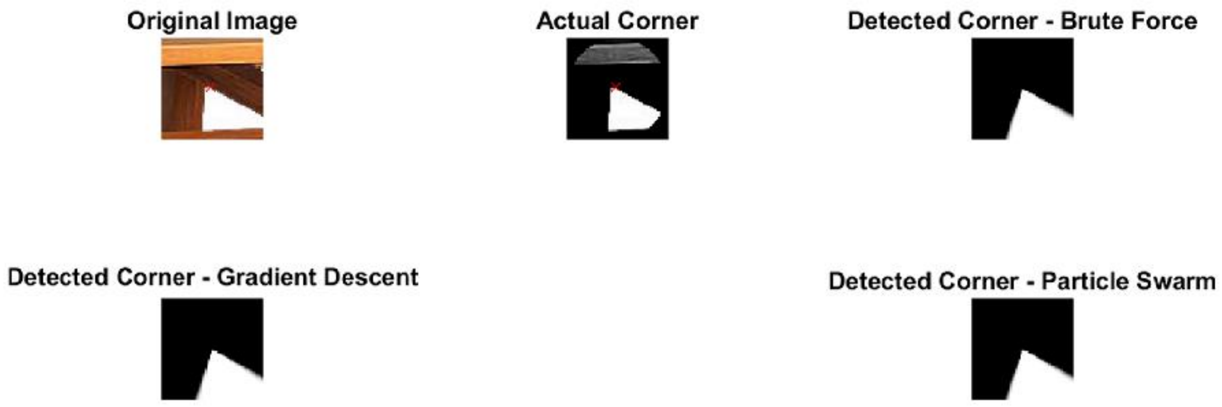
Original Image
Actual Corner
Detected Corner - Brute Force
Detected Corner - Gradient Descent
Detected Corner - Particle Swarm

Figure 4.12: Graphical Comparison of Optimization Methods on Stool Image

4.3.3 Example 3 – Corners in Cameraman Image

In this example, corners were found in the image of the cameraman. Figure 4.13 shows the corner in the original image as detected by Harris corner detection, as well as the detected corner orientations for each method. From this figure, it is evident that all of the optimization methods converged to the same solution on this corner from a real image. In this case, it would be very difficult for a human to determine that this point is a corner. However, since there is a larger mass of white pixels on the left side of the image, the orientation estimator correctly determined that the corner was oriented in this direction.

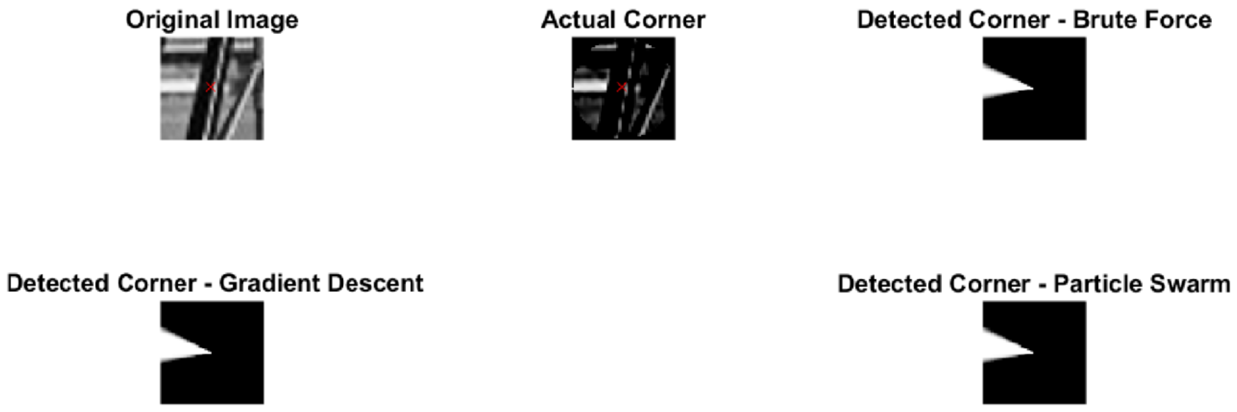
Original Image
Actual Corner
Detected Corner - Brute Force
Detected Corner - Gradient Descent
Detected Corner - Particle Swarm

Figure 4.13: Graphical Comparison of Optimization Methods on Cameraman Image

CHAPTER 5

CONCLUDING REMARKS AND FUTURE WORK

5.1 Concluding Remarks

The research presented in the thesis provides a practical method for estimating the orientation of corners in real images. The journal paper by Muhlich, Friedrich, and Aach presented the theory of efficiently implementing multisteerable filters in the Fourier domain, but did not show examples of how a corner orientation estimator could be implemented [9]. This research focuses on efficiently implementing a multisteerable filter for corner orientation detection. In the conclusion of the work by Muhlich, Friedrich, and Aach, they mentioned that further research could be done on finding efficient optimization methods for finding the global maximum in the similarity function [9]. This research aimed to fill in the missing information with regards to efficiently solving for the global maximum in the similarity function for corners.

To increase the speed of determining the orientation angle of a corner, several methods were proposed in this research. The first method involving storing the coefficients in memory. Since the coefficients are much smaller than an image, they can easily be stored in memory and loaded before attempting to determine the orientation of a corner. Storing the coefficients in this way sped up the performance of the corner detection algorithm by around 3 times. However, it was noted that further improvements might by implementing the similarity function in parallel. For particle swarm method, the calculation of the similarity function for each particle can be done in parallel, which should greatly increase the speed of this method. However, the gradient descent method cannot be implemented in parallel because only one point is tested at a time, and the gradient must first be calculated to determine the next point to test.

Several different optimization methods were tested in this research, which include Levenberg-Marquardt optimization, gradient descent optimization, and particle swarm optimization. It was determined that the Levenberg-Marquardt optimization method was not well suited for optimizing the type of similarity function characteristic of corner features. Muhlich, Friedrich, and Aach used Levenberg-Marquardt optimization to solve for the global optimum in the similarity function; however, none of their examples involved corners and their similarity functions were obtained with only one orientation angle [9].

The gradient descent method performed well on synthesized ideal corners, but took many more iterations to converge than the particle swarm method. Moreover, when noise was present, as is typical in real images, the gradient descent method had a much larger probability of converging to a local maximum. It was shown that the presence of noise in an image can directly cause more local optimum to appear in the similarity function. The particle swarm method, however, performed much better by not converging to a local maximum except in the case of speckle noise. However, even when speckle noise was present in an image, the particle swarm method only converged to a local maximum 7% of the time, while the gradient descent method converged to a local maximum over 50% of the time.

Finally, both optimization methods were tested on corners from real images. It was shown that all of the optimization methods converged to the global maximum for the examples shown in this thesis; however, the particle swarm method still proved to be superior to the gradient descent method in terms of number of iterations required to converge. One important note from corners in real images is that they do not always appear to be corners to a human being. Moreover, it is often difficult to distinguish by eye the orientation of a corner. However, it is not very important the orientation of the angle be determined with extreme accuracy. What is most important for most computer vision applications is that a method of determining the orientation of corners remains consistent under various homography transformations, and will not converge to local maximum. For this reason, the particle swarm optimization method described in this thesis provides a good framework for comparing orientation angles of corners in images.

5.2 Future Work

For future work, other optimization methods could be tested, or modifications to the particle swarm method can be implemented to result in greater accuracy for determining the orientation angles of corners. In the work by Muhlich, Friedrich, and Aach, they proposed windowing the coefficients with a window such as the Hamming window in order to reduce oscillations [9]. However, oscillations are similar to noise, and particle swarm optimization method proved to be quite robust to noisy conditions, so a windowing function may not be necessary. Furthermore, windowing the coefficients in this way reduces the sharpness of the edges that make up the corner, which makes it difficult for there to be extremely high precision in determining the orientation angle. Further research could be done on implementing multisteerable filters for corners without the use of windowing function in order to increase the accuracy of the orientation estimator. Lastly, further improvements could be made by exploring a parallel implementation of multisteerable filters using the graphics processing unit.

BIBLIOGRAPHY

[1] D. Tamrakar and P. Khanna, "Multispectral palmprint recognition using steerable filter," *2014 9th International Conference on Industrial and Information Systems (ICIIS)*, Gwalior, 2014, pp. 1-5.

[2] Qi Li, Isao Sato and Yutaka Murakami, "Steerable filter based multiscale registration method for JERS-1 SAR and ASTER images," *2007 IEEE International Geoscience and Remote Sensing Symposium*, Barcelona, 2007, pp. 381-384.

[3] E. Shang, J. Li, X. An and H. He, "Lane Detection Using Steerable Filters and FPGA-based Implementation," *Image and Graphics (ICIG), 2011 Sixth International Conference on*, Hefei, Anhui, 2011, pp. 908-913.

[4] S. Xu, Y. Jia and X. Zhang, "Coding Facial Expression with Oriented Steerable Filters," *2006 International Conference on Image Processing*, Atlanta, GA, 2006, pp. 2057-2060.

[5] D. Alexiadis, N. Mitianoudis and T. Stathaki, "Multidimensional steerable filters and 3D flow estimation," *2014 IEEE International Conference on Image Processing (ICIP)*, Paris, 2014, pp. 2012-2016.

[6] Zhou Weiping and Shu Huazhong, "Detection of Cerebral Vessels in MRA Using 3D Steerable Filters," *2005 IEEE Engineering in Medicine and Biology 27th Annual Conference*, Shanghai, 2005, pp. 3249-3252.

[7] A. Ozmen and E. Tufan Akman, "Edge detection using steerable filters and CNN," *Signal Processing Conference, 2002 11th European*, Toulouse, 2002, pp. 1-4.

[8] S. Anand and H. Abirami, "Non-separable steerable filters for edge detection," *Signal Processing, Communication, Computing and Networking Technologies (ICSCCN), 2011 International Conference on*, Thuckafay, 2011, pp. 665-669.

[9] M. Mühlich, D. Friedrich and T. Aach, "Design and Implementation of Multisteerable Matched Filters," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 2, pp. 279-291, Feb. 2012.

[10] F. David, J. Ponce. *Computer Vision: A Modern Approach*. Boston: Pearson, 2012.

[11] Z. Ye, Y. Pei and J. Shi, "An Improved Algorithm for Harris Corner Detection," *Image and Signal Processing, 2009. CISP '09. 2nd International Congress on*, Tianjin, 2009, pp. 1-4.

[12] H. Jiang, T. P. Tian and S. Sclaroff, "Scale and rotation invariant matching using linearly augmented trees," *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, Providence, RI, 2011, pp. 2473-2480.

[13] M. Jacob and M. Unser, "Design of steerable filters for feature detection using canny-like criteria," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 8, pp. 1007-1019, Aug. 2004.

[14] W. T. Freeman and E. H. Adelson, "The design and use of steerable filters," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 9, pp. 891-906, Sep 1991.

[15] J. Bigun, G.H. Granlund, "Optimal Orientation Detection of Linear Symmetry," *Proc. IEEE First Int'l Conf. Computer Vision*, pp. 433-438, June 1987.

[16] J. Bigun, G. H. Granlund and J. Wiklund, "Multidimensional orientation estimation with applications to texture analysis and optical flow," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 8, pp. 775-790, Aug 1991.

[17] M. Kass and A. Witkin, "Analyzing Oriented Patterns," *Computer Vision, Graphics, and Image Processing,* vol. 37, pp. 362-385, 1987

[18] S. Di Zenzo, "A Note on the Gradient of a Multi-Image," *Computer Vision, Graphics, and Image Processing*, vol. 33, pp. 116-125, 1986.

[19] M. Shizawa and K. Mase, "Simultaneous multiple optical flow estimation," *Pattern Recognition, 1990. Proceedings., 10th International Conference on*, Atlantic City, NJ, 1990, pp. 274-278 vol.1.

[20] M. Shizawa and K. Maze, "Unified computational theory for motion transparency and motion boundaries based on eigenenergy analysis," *Computer Vision and Pattern Recognition, 1991. Proceedings CVPR '91., IEEE Computer Society Conference on*, Maui, HI, 1991, pp. 289-295.

[21] T. Iso and M. Shizawa, "Detecting L-, T-, and X-Junctions from Low-Order Image Derivatives," *Proc. Visual Comm. And Image Processing*, pp. 1185-1197, Nov. 1993.

[22] T. Aach, C. Mota, I. Stuke, M. Muhlich and E. Barth, "Analysis of Superimposed Oriented Patterns," in *IEEE Transactions on Image Processing*, vol. 15, no. 12, pp. 3690-3700, Dec. 2006.

[23] M. Mulich and T. Aach, "A Theory for Multiple Orientation Estimation," *Proc. European Conf. Computer Vision*, H. Bischof and A. Leonardis, eds., pp. 69-82, 2006.

[24] M. Muhlich and T. Aach, "Analysis of Multiple Orientations," in *IEEE Transactions on Image Processing*, vol. 18, no. 7, pp. 1424-1437, July 2009.

VITA

The author was born in 1991 in New Orleans, Louisiana. He completed his Bachelors of Science in Electrical Engineering from the University of New Orleans in 2013. He is expected to finish his Masters in Electrical Engineering from the University of New Orleans in August 2016. His areas of interests are computer vision and power system protection.