ABSTRACT

KHANNA, SANJEEV. Developing an Interactive, GUI Based, Cross Platform Image Processing, Editing, and Algorithm Evaluation Tool. (Under the direction of Dr. Wesley Snyder.)

There is a strong need for a software toolkit which serves to strike the balance between algorithm evaluation and image manipulation. SKIPT (Snyder Khanna Image Processing Toolkit) is a simple, easy to use, interactive, and cross platform image processing and algorithm evaluation program with the motive to serve as a valuable teaching tool in understanding the fundamental computer vision principles/algorithms. It is an application that has minimum dependencies on external libraries, and uses as few libraries as possible.

We demonstrate that merely by using Qt and IFS (Image File System) libraries, it is possible to realize a simple yet a comprehensive and powerful image processing toolkit.
© Copyright 2014 by Sanjeev Khanna
All Rights Reserved
Developing an Interactive, GUI Based, Cross Platform Image Processing, Editing, and Algorithm Evaluation Tool

by
Sanjeev Khanna

A thesis submitted to the Graduate Faculty of North Carolina State University in partial fulfillment of the requirements for the Degree of Master of Science

Electrical Engineering
Raleigh, North Carolina
2014

APPROVED BY:

______________________________  __________________________
Dr. Edgar Lobaton                Dr. Edward Grant

______________________________
Dr. Wesley Snyder
Chair of Advisory Committee
DEDICATION

To my parents.
BIOGRAPHY

The author was born in Delhi, India in June 1990. After completing his high school from Apeejay School Noida, he went to JSS Academy of Technical Education, Noida to pursue a bachelor's degree in Instrumentation and Control Engineering in 2008.

In August 2012, he moved to Raleigh, North Carolina and began his graduate school as a master's student in Electrical Engineering at North Carolina State University. An avid researcher in the Robotics and Computer Vision, he aspires to pursue a career in Object Recognition, Human-Machine Interaction and Machine Learning with emphasis on designing smart Robotic Systems.
ACKNOWLEDGEMENTS

I cannot express my gratitude towards my advisor and my idol Dr. Wesley Snyder who has guided me with immense patience and sincere dedication every single day during my thesis. He believed in me and mentored me when no one else would have. He has been more like a family member, a father figure. It is only because of his constant efforts and hard work that I have managed to evolve from a student to a researcher. Without him, this thesis would not have been possible. Thanks a million Dr. Snyder, you are the best.

A big thanks to my committee members - Dr. Edgar Lobaton and Dr. Edward Grant for their guidance and valuable inputs. They were always there to answer my smallest of queries.

I would also like to thank my parents for their constant encouragement and support. They have always been there for me. A special mention to my colleague and friend Benjamin Riggan for helping me greatly and often spending hours with me debugging codes. Finally, I would like to acknowledge my roommate Varun Khurana for being exceptionally understanding and transforming my stress moments to our crazy fun times.
# TABLE OF CONTENTS

## LIST OF TABLES

<table>
<thead>
<tr>
<th>Table of Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>viii</td>
<td></td>
</tr>
</tbody>
</table>

## LIST OF FIGURES

<table>
<thead>
<tr>
<th>List of Tables</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ix</td>
<td></td>
</tr>
</tbody>
</table>

## Chapter 1 Introduction

1.1 Motivation .................................................. 1
1.2 Select Features ............................................. 2
1.3 Thesis Organization ......................................... 3

## Chapter 2 Supported Algorithms

2.1 Image Pre-Processing ........................................ 5
  2.1.1 Histogram Equalization .................................. 5
  2.1.2 Window and Level .......................................... 7
  2.1.3 Dilation ................................................... 7
  2.1.4 Erosion .................................................... 8
  2.1.5 Closing .................................................... 8
  2.1.6 Opening .................................................... 8
  2.1.7 Brightness ............................................... 9
  2.1.8 Warmness ................................................. 9
  2.1.9 Coolness ................................................ 9
  2.1.10 Smoothing/Blur ......................................... 9
  2.1.11 Sharpen .................................................. 11
  2.1.12 Grayscale .............................................. 12
  2.1.13 Inverted Grayscale ................................... 12
  2.1.14 Log ..................................................... 12

2.2 Edge Detection Algorithms .................................. 13
  2.2.1 Canny .................................................... 13

2.3 Image Derivatives ............................................ 16
  2.3.1 First Derivative X Direction, First Derivative Y Direction, Second Derivative X Direction, Second Derivative Y Direction ............. 16
  2.3.2 First Derivative X Direction Gaussian, First Derivative Y Direction Gaussian, Second Derivative X Direction Gaussian, Second Derivative Y Direction Gaussian ......................... 16
  2.3.3 First Order Derivative Operators Used in Edge Detection .......... 16
  2.3.4 Second Order Derivative Operators Used in Edge Detection .......... 19

2.4 Feature Detectors ............................................. 22
  2.4.1 The Harris Detector ...................................... 23
  2.4.2 The Harris Laplace Detector .............................. 24
  2.4.3 The SIFT Detector ....................................... 25
2.5 Image Segmentation .......................................................... 26
  2.5.1 Convex Hull ............................................................. 28
  2.5.2 Active Contours/Snakes ............................................. 30
  2.5.3 Color Based Segmentation ........................................... 31
  2.5.4 Watershed Segmentation ............................................. 32
2.6 Miscellaneous Operations ................................................. 33
  2.6.1 Line Profile ............................................................ 33
  2.6.2 Image Histogram and associated Image Statistics ............... 34
  2.6.3 Acquiring pixel intensities from user selected points .......... 35

Chapter 3 Comparison With Similar Existing Software ..................... 36
  3.1 MATLAB/Simulink ...................................................... 36
  3.2 Wolfram Mathematica .................................................. 37
  3.3 ImageNets ............................................................... 37
  3.4 Computer Vision and Image Processing Tools ...................... 38
  3.5 GNU Image Manipulation Program ................................... 39
  3.6 Other Image Processing/Image Viewing Software Packages ........ 39

Chapter 4 Operational Overview ............................................. 41
  4.1 High Level Operation .................................................. 41
  4.2 Low Level Operation ................................................... 47

Chapter 5 System Design and Analysis ....................................... 51
  5.1 Why prefer Qt over other IDEs with GUI builders? ................ 51
  5.2 System Performance .................................................... 52

Chapter 6 Future Work and Conclusions ...................................... 56
  6.1 Future Work ............................................................. 56
  6.2 Conclusions ............................................................. 62

BIBLIOGRAPHY .................................................................. 63

APPENDIX ........................................................................ 67
Appendix A SKIPT User Guide .................................................. 68
  A.1 Features ................................................................. 68
  A.2 Installation and Running Instructions ................................. 70
  A.3 Working with SKIPT ................................................... 70
    A.3.1 File Menu ........................................................... 71
    A.3.2 Image Pre-Processing Menu ................................... 73
    A.3.3 Zoom Menu ......................................................... 78
    A.3.4 Derivatives Menu ................................................ 78
    A.3.5 Edge Detectors Menu ............................................ 81
A.3.6 Feature Detectors Menu ........................................ 84
A.3.7 Image Segmentation Menu ...................................... 85
A.3.8 Miscellaneous Operations Menu ............................... 87
LIST OF TABLES

Table 3.1  Comparison of various softwares  ......................... 40
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Structuring Elements</td>
<td>8</td>
</tr>
<tr>
<td>2.2</td>
<td>A 3 x 3 smoothing kernel used in Homogeneous Smoothing</td>
<td>10</td>
</tr>
<tr>
<td>2.3</td>
<td>Gaussian Kernel</td>
<td>11</td>
</tr>
<tr>
<td>2.4</td>
<td>Sharpening Filters</td>
<td>12</td>
</tr>
<tr>
<td>2.5</td>
<td>Discrete approximation to a Gaussian function with $\sigma = 1.4$</td>
<td>14</td>
</tr>
<tr>
<td>2.6</td>
<td>[Gre] Orientation Assignment</td>
<td>15</td>
</tr>
<tr>
<td>2.7</td>
<td>Edge Detection Using Sobel Operators</td>
<td>17</td>
</tr>
<tr>
<td>2.8</td>
<td>Edge Detection Using Prewitt Operators</td>
<td>18</td>
</tr>
<tr>
<td>2.9</td>
<td>Edge Detection Using Robert-Cross Operators</td>
<td>18</td>
</tr>
<tr>
<td>2.10</td>
<td>Laplacian Kernels</td>
<td>19</td>
</tr>
<tr>
<td>2.11</td>
<td>A 5x5 Laplacian Kernel</td>
<td>20</td>
</tr>
<tr>
<td>2.12</td>
<td>[Wan] Gaussian (red), Gaussian First Derivative (green), and Gaussian Second Derivative (blue)</td>
<td>21</td>
</tr>
<tr>
<td>2.13</td>
<td>[Wan] Plot of Laplacian of Gaussian</td>
<td>21</td>
</tr>
<tr>
<td>2.14</td>
<td>A 5x5 LoG kernel</td>
<td>22</td>
</tr>
<tr>
<td>2.15</td>
<td>[BVS10] Classification of Feature Detection Methods</td>
<td>23</td>
</tr>
<tr>
<td>2.16</td>
<td>“The Harris detector searches for image neighborhoods where the second-moment matrix $C$ has two large eigenvalues, corresponding to two dominant orientations. The resulting points often correspond to corner-like structures” [GL11].</td>
<td>24</td>
</tr>
<tr>
<td>2.17</td>
<td>Difference of Gaussians in SIFT [Low99]</td>
<td>27</td>
</tr>
<tr>
<td>2.18</td>
<td>Keypoints detection in SIFT [Low99]</td>
<td>27</td>
</tr>
<tr>
<td>2.19</td>
<td>[YL] Illustrating the stages between oversegmentation (extreme left) and undersegmentation (extreme right)</td>
<td>28</td>
</tr>
<tr>
<td>2.20</td>
<td>[Ala] The convex hull ideology</td>
<td>29</td>
</tr>
<tr>
<td>2.21</td>
<td>[BMS97] Example of a snake. The sequence of images shows the evolution of the snake from a user-defined original position to its final state</td>
<td>31</td>
</tr>
<tr>
<td>2.22</td>
<td>[ZMLW09] The concept of watershed</td>
<td>32</td>
</tr>
<tr>
<td>2.23</td>
<td>“Example of edgeness watershed segmentation. Top left: original noisy image. Top right: edgeness image. Bottom left: landscape version of the edgeness image. Dams will be formed when the water level reaches the tops of the barricade. An uneven barricade height will not disturb the formation of the correct dam marking an object boundary. Bottom right: all dams formed; the watershed of the edgeness image.” [Uni]</td>
<td>33</td>
</tr>
<tr>
<td>2.24</td>
<td>[C.T] Illustrating Line Profile</td>
<td>34</td>
</tr>
<tr>
<td>4.1</td>
<td>SKIPT Working Environment</td>
<td>42</td>
</tr>
</tbody>
</table>
Figure 4.2 Image Selection .......................................................... 42
Figure 4.3 Brightness operation in SKIPT. The user with the help of the slider can vary of amount of brightness/darkness desired in the image. ........ 43
Figure 4.4 Window and Level Operation ........................................... 43
Figure 4.5 First Derivative X Direction Operation ............................... 44
Figure 4.6 First Derivative Y Direction Gaussian Operation. The slider allows the user to vary the value of sigma used for calculating the derivative. .... 44
Figure 4.7 Edge Detection using Sobel Operator ................................. 45
Figure 4.8 Canny Edge Detection Operation ...................................... 45
Figure 4.9 Prototype of the SKIPT Block Diagram Environment ............. 47
Figure 4.10 Sequence of steps followed during an image operation in SKIPT ... 50
Figure 5.1 Brightening an image first and increasing its coolness later ...... 53
Figure 5.2 Cooling an image first and increasing its brightness later .......... 54
Figure 6.1 Proposed SKIPT Block Diagram Environment ....................... 57
Figure 6.2 Pre-Processing Operations ........................................... 58
Figure 6.3 Edge Detection Operations ........................................... 59
Figure 6.4 Feature Detection Operations ........................................ 59
Figure 6.5 Derivatives Operations ............................................... 60
Figure 6.6 Arithmetic Operations ............................................... 60
Figure 6.7 Program Flow Operations ........................................... 61
Figure 6.8 Segmentation Operations ............................................. 61
Figure A.1 SKIPT Working Environment .......................................... 71
Figure A.2 Image loading and image selection for further operations .......... 72
Figure A.3 Prototype of the SKIPT Block Diagram Environment .............. 73
Figure A.4 Histogram Equalization operation in action. From the Image Pre-Processing Menu, the user clicks on Histogram Equalization and obtains the equalized image shown in the child window. ......................... 74
Figure A.5 Window and Level Operation. Pixel values darker than the lower threshold value are mapped to black (value 0), while pixels brighter than the upper threshold value are mapped to white (value 255). All pixel values in between the two thresholds are scaled linearly in 0-255 range. ......... 75
Figure A.6 Brightness operation in action. The slider is used to vary the amount of lightness/darkness in the image. Warmness and Coolness operations are also similar. .................................................. 76
Figure A.7 Smoothing operation in action. The slider is used to change the value of Gaussian Sigma, varying the amount of blurring in the image. ......... 77
Figure A.8 Grayscale operation in action. From the Image Pre-Processing Menu, the user clicks on Grayscale and obtains the monotone image shown in the child window. Inverted Grayscale and Log are also similar. ......... 78
| Figure A.9 | First Derivative X Direction Operation | 79 |
| Figure A.10 | First Derivative Y Direction Gaussian Operation. The slider allows the user to vary the value of sigma used for calculating the derivative. | 80 |
| Figure A.11 | Edge Detection using Sobel Operator. The slider is used to vary the Threshold value. | 82 |
| Figure A.12 | Canny Edge Detection Operation. It uses two thresholds. Because of noise, using a single threshold has its limitations. A high threshold removes significant information, while a low threshold leads to generating numerable false edge points. To avoid this, 2 thresholds - (T_{upper}) and (T_{lower}) are used. Not only does this reduces false edge points in the output image, but also links the edges and reduces discontinuities. | 84 |
| Figure A.13 | Harris Laplace Operation in action. There are three user-defined inputs - sigma, threshold value for Harris points, and threshold value for Laplacian scale space. The feature points along with their corresponding scale are listed in a text file. | 85 |
| Figure A.14 | Color Based Segmentation in SKIPT. A reference color is taken from which Cartesian distances are taken at every pixel color. This distance is then thresholded. For instance, if the user wants to segment yellow colors in the image then the red channel and green channel are made high. The thresholding value is correspondingly adjusted so as to bring the region of interest inside the 'sphere in RGB space', centered on the reference color. | 87 |
Chapter 1

Introduction

1.1 Motivation

There is a strong need for a software toolkit which serves to strike the balance between algorithm evaluation and image manipulation. On one hand there is MATLAB which is consistently rated by the industry as the best algorithm evaluation program. On the other, there is Adobe Photoshop which gives an interactive artistic manipulation interface. In between the shores of algorithm evaluation programs and image manipulation programs there is an ocean of image editing programs, each with its own set of advantages and disadvantages.

MATLAB through its rich toolbox support, is capable of providing nearly every functionality. It does however requires the user to manually write a code and lacks the ‘plug-and-play’ functionality. Photoshop, allows the user to change image characteristics by varying parameters through a rich graphical user interface (GUI). There is no need to write long codes and everything happens in background. However, Photoshop is strictly a graphics manipulation program. The user cannot evaluate the performance of image processing algorithms.

Wouldn't it be nice to have a software having the algorithm evaluation functionality, with an interactive interface, and without the need of writing any codes? Wouldn't it be convenient to have a software giving us the best of both worlds - MATLAB and Photoshop? Well, now there is. **SKIPT is the answer.**

SKIPT (Snyder Khanna Image Processing Toolkit) is a simple, easy to use, interactive,
and cross platform image processing and algorithm evaluation program with the motive to serve as a valuable teaching tool in understanding the fundamental computer vision principles/algorithms. From raw images to popular formats - jpg, jpeg, png, xpm, it supports images with pixels of any data type (complex, double, float, int) and any number of dimensions. SKIPT is a basic yet powerful software intended for students and researchers wishing to delve deeper in image processing/object recognition field. Through its ability to compare the performance measures and working of algorithms by simply varying parameters with a GUI and without the need to actually write the code, it becomes an effective learning medium. The idea is to give users a visual feel, comparison, and an active understanding of essential inter-class and intra-class computer vision algorithms. Even for computer vision veterans, there are several features making SKIPT a useful pre-processing or in-processing tool.

1.2 Select Features

The software is written in C and C++ and developed on the Qt application framework. Qt is a popular cross-platform developing environment for developing GUI application software. Some of the features of SKIPT are:

- Platform independence and native look on every supported platform
- Ability to switch to a block diagram based image processing, algorithm simulation, and evaluation environment
- Code generation and export
- Ability to switch operations on original image and current images
- Support of many image formats (Read/Write - IFS, BMP, JPG, JPEG, PNG, PPM, XBM, XPM; Read - GIF, PBM, PGM)
- Support of images with pixels of any data type - unsigned 8 bit, signed 8 bit, unsigned 16 bit, signed 16 bit, 32 bit integer, 32 bit floating point, 64 bit double, complex short, complex double, complex float
Popular segmentation algorithms evaluation - Active Contours (Snakes)*, Convex Hull*, Watershed, Color Thresholding

Popular edge detection techniques along with comparison - Sobel, Prewitt, Robert Cross, Canny, Laplacian, Laplacian of Gaussian

Popular feature detection algorithms evaluation along with comparison - Harris, Harris Laplace, SIFT*


Line Profile - both grayscale and colored images

Image Histogram with statistics - mean, standard deviation, max pixel value, min pixel value

Image pre-processing and basic editing - Histogram Equalization, Window and Level, Erosion*, Dilation*, Brightness, Warmness, Coolness, Blur, Sharpen, Rotate

Saving images

Printing images

Keyboard Shortcuts

Comparing images side by side

Acquiring pixel intensities at user selected points

* denotes that the feature/operation has not yet been implemented.

1.3 Thesis Organization

The thesis report starts with Chapter 2, briefly explaining the supported algorithms along with the methodologies used to implement them. Chapter 3 describes the overall working of SKIPT in detail. Chapter 4 compares SKIPT with the existing software, highlighting the distinctness and salient features of SKIPT. The system design and analysis of the application
have been covered in Chapter 5. Finally, the report ends with Chapter 6 presenting the conclusion and future work. It also lists recommendations for expanding existing functionality.
Chapter 2

Supported Algorithms

This chapter describes the algorithms supported by SKIPT. These algorithms are divided into 6 categories according to type, nature, and use - Image Pre-Processing, Edge Detectors, Image Derivatives, Feature Detectors, Image Segmentation, and Miscellaneous Operations. Such segregation makes it easy to find the algorithm needed. The block diagram version of SKIPT inherits selected algorithms from the regular version. They are mentioned in the respective section.

2.1 Image Pre-Processing

Image pre-processing involves operations to enhance image data and/or or suppress undesired distortions prior to any computational processing. These operations do not increase the image information, yet they come with a risk of emphasizing image artifacts and loss of information if not used correctly.

2.1.1 Histogram Equalization

Histogram equalization [FPWW] is a method for adjusting and stretching image intensity range to enhance contrast. It uses a monotonic, non-linear mapping to change the intensity values of the input image's pixels in such a way that the output image's pixels have a uniformity in the distribution of intensities.

Let there be a given image $f$ with integer pixel intensities ranging from 0 to $L - 1$. Here $L$ corresponds to the total number of intensity values (usually 256). The probability of
occurrence of a pixel with intensity $i$ will be given as

$$p(x_i) = \frac{\text{number of pixels with intensity } i}{\text{total number of pixels}} \quad 0 \leq i \leq L - 1$$

Note that $p(x_i)$ is essentially the image histogram normalized to integrate to 1.

The cumulative distribution function (CDF) for intensity $i$ is the sum of all probabilities $p(x) \forall x < x_i$. Thus

$$\text{CDF}(x_i) = \sum_{k=0}^{i} p(x_k)$$

The CDF is the accumulated normalized image histogram. The overall objective is to obtain a uniform distribution of intensities, i.e. an image with a linear CDF. Thus, the histogram equalized image $H_i$ is given as

$$H_i = \text{floor}((L - 1) \sum_{k=0}^{i} p(x_k))$$

where floor() rounds down to the nearest integer. A more accurate approximation can be achieved by using the transformation

$$H_i = \text{floor}((L - 1) \sum_{k=0}^{i} p(x_k) + 0.5)$$

The values obtained correspond to a normalized equalized histogram. They have to be mapped to their original range. A simple transformation shown below does the mapping

$$H_{i_{\text{mapped}}} = H_i \times \left( \frac{\max[f(x_i)] - \min[f(x_i)]}{255} \right) + \min[f(x_i)]$$

where $\max[f(x_i)]$ and $\min[f(x_i)]$ corresponds to the maximum and minimum pixel frequencies of the original image.
2.1.2 Window and Level

Window and Level [Pub] provides contrast expansion of image pixels within a given window range. There are two parameters which control the pixel intensities - the Window which is the width of the range, and the Level which is the mid-gray brightness value of the window. Pixel values darker than the lower threshold limit are mapped to black (value 0), while pixels brighter than the upper threshold limit are mapped to white (value 255). All pixel values in between the two thresholds are scaled linearly in 0-255 range according to the slope of the window. The maximum limit gets updated depending upon the bit depth of image. For example if the image depth is 8bit then the maximum limit becomes 255, while for 16bit images the maximum limit becomes 65535.

2.1.3 Dilation

Dilation and Erosion are the fundamental Morphological Operations. Image morphology can be described as study of image shapes. The idea is to examine every pixel of an image with a predefined shape of pixels and bringing about a change in the image. Depending upon the manner in which this 'shaped element' interacts with the image pixel's neighborhood, the result of the operation is determined.

Morphological Image Processing is a simple and useful tool for object segmentation based on shape. It can also be used as a noise filtering agent to improve the quality of the image.

The fundamental instrument in image morphology is the structuring element (SE) - an arrangement of pixels representing a shape and having a defined origin. There are no shape constraints or shape definitions for the SE, however a simple shape having the same width and height (such as a square) is usually used for efficiency and to have a uniformity in up/down scaling.

The dimensions of the matrix used to construct the SE specify its size, while the pattern formed by the arrangement of ones and zeros represent the shape of the structuring element [Auc]. Some random structuring elements are described in figure 2.1.

In a sense, a SE in the context of morphological image processing is analogous to convolution kernel in the context of linear image filtering [Auc]. Consider a SE placed over binary image. If for all the entries of the SE set to 1, the corresponding image pixels also have a high value (value 1), then the SE fits the image. Conversely, a SE hits an image when
atleast one of its elements overlaps with a high value of image pixel.

The dilation of a binary image ‘f’ by a SE ‘s’ is denoted $f \oplus s$. It produces a new binary image with the shape of SE added in all those locations $(x, y)$ where the SE hits the input image ‘f’. Dilation leads to addition of pixels at the object boundaries.

### 2.1.4 Erosion

The erosion of a binary image ‘f’ by a SE ‘s’ is denoted $f \ominus s$. It produces a new binary image with the shape of SE subtracted from all those locations $(x, y)$ where the SE fits the input image ‘f’.

Erosion has the opposite effect as that of dilation. It subtracts (reduces) pixels from the boundaries of the object.

Although Dilation and Erosion are not inverses of each other, dilating the ‘foreground’ is the same as eroding the ‘background’. Similarly, eroding the ‘foreground’ is the same as dilating the ‘background’.

### 2.1.5 Closing

Dilation of an image followed by erosion is closing. It is denoted by $(f \oplus s) \ominus s$. Closing can fill the spaces between the image regions and also maintains the initial shape of the image. The dilation part fills the spaces while erosion part ensures the same image shape.

### 2.1.6 Opening

Erosion of an image followed by dilation is opening. It is denoted by $(f \ominus s) \oplus s$. Since the operation actually ‘opens’ up gaps between objects connected by pixels, it is called Opening.
It also removes sharp areas/edges sticking out of regions.

Opening and Closing are dual operations of each other and not INVERSES. An opening followed by a closing will NOT restore the original image.

### 2.1.7 Brightness

Image brightness [Cha] is the degree of lightness (color closer to white) or darkness (color closer to black) in the image. Low brightness makes the tones darker, while high brightness makes the tones lighter [Webc]. Brightness differs from contrast in the sense that contrast emphasizes or de-emphasizes the difference between the lighter and darker regions in an image.

To increase the brightness we simply increment the value of 3 channels (RGB). Conversely, to decrease the brightness we decrement the value of 3 channels.

### 2.1.8 Warmness

Warmness refers to the image colors closeness to yellow. To make an image warm we increase the yellow channel, i.e. increase the red and green channel. The blue channel remains the same.

### 2.1.9 Coolness

Coolness refers to the image colors closeness to blue. To make an image cool we increase the blue channel by some amount, leaving the red and green channel unchanged.

### 2.1.10 Smoothing/Blur

Smoothing [Fer] [Webf] consists of convolving the image with a blurring kernel. The intent is to smooth the image and make it appear 'blurry'. As a result, the hard edges in the image are softened, and the overall spatial frequency is lowered.

There are various ways to smooth or blur an image. Most common techniques are the following:

1. Homogeneous Smoothing
   The simplest smoothing technique. Every pixel of the image is assigned the average value of its neighboring pixels. The choice of the kernel size decides the neighborhood
to be considered. If the kernel is too large then the intricate features of the image may disappear and image will look blurred. If the kernel is too small, then the noise artifacts will still be present.

2. Gaussian Smoothing

Most commonly used smoothing technique. A Gaussian kernel is traversed across the image pixel by pixel to produce the smoothed image. If we take the kernel described in figure 2.3, the brightness value of the center pixel along with its 24 neighbors is multiplied with the corresponding weights of the kernel and added together. The result is divided by the normalizing term and the value obtained becomes the new pixel brightness value. Gaussian Smoothing has been implemented in SKIPT.

3. Median Smoothing

A median kernel is convolved with the image, and every pixel of the image is assigned the median value of its neighboring pixels. Similar to Homogeneous Smoothing, the choice of the kernel size decides the neighborhood to be considered.

4. Bilateral Smoothing

Most advanced technique used in image smoothing. Every pixel of the image is assigned the weighted average value of its neighboring pixels. A Gaussian distribution can serve to be this weight. This is the only technique capable of reducing noise in the image and at the same time maintaining the integrity of the edges. However, this added advantage comes at the expense of increased processing time.

The primary difference between the various smoothing methods is the choice of kernel used for convolution. For example, the kernel used in homogeneous smoothing has the same values across all its rows and columns. A 3 x 3 homogeneous smoothing kernel is shown in figure 2.2.

\[
\frac{1}{9} \begin{bmatrix}
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1 \\
\end{bmatrix}
\]

**Figure 2.2** A 3 x 3 smoothing kernel used in Homogeneous Smoothing
A $5 \times 5$ Gaussian kernel can be of the type shown in figure 2.3. Note that the origin of the kernel is in the center. A Gaussian kernel is isotropic and the weights are spread evenly in all directions from the center. Weight at the center has more value and the weights of the neighbors decreases on increasing the distance from the center.

Some important facts about smoothing kernels (filters)

- Number of rows and number of columns of a kernel should be odd
- Processing time increases upon increasing the ‘size’ of the kernel. In this context, ‘size’ refers to the radius of the kernel. For instance, in figure 2.3, the size of the kernel is 2.

2.1.11 Sharpen

“Sharpness can be defined as edge contrast, that is the contrast along the edges in a picture” [Weba]. The idea is to highlight details in an image without introducing noise or artifacts. Increasing sharpness increases the contrast only near the edges in the photo, and the other areas of the image are remained unaffected [Weba]. As a result, the object boundaries appear crisp, not gradual.

The simplest way to sharpen an image is to pass a sharpening filter over every pixel in the image. Typical sharpening filters are shown in figure 2.4 [Fou].

In essence, a sharpening filter is a high-pass filter - a filter which brings out the fine details in the image. The negative signs in the kernel ensures that for the adjacent pixels there is no change in the image for constant intensities. For the case of one pixel being brighter than its immediate neighbors, the pixel intensity gets increased. However, high-pass filtering amplifies noise. If the noise in the original image is small, the effect is not that prominent. But for images having considerable amount of noise, further noise amplifica-
tions considerably degrades the image. High-pass filtering also causes small details to be distinctly visible and stand out in the image. While it may be useful in certain situations, but usually an over-sharpened image tends to be coarse and appears unnatural. For instance, a black dot might appear like a blob in an over-sharpened image [Webb].

2.1.12 Grayscale

A grayscale image is an image in which the value of each pixel is a single sample, that is, it carries only intensity information. In other words an image in which the red, green and blue components all have equal intensity in RGB space. Greyscale images are also called monochromatic, denoting the presence of only one color.

2.1.13 Inverted Grayscale

The complement of a grayscale image is an inverted grayscale image. For instance, if there is a 16bit image, the inverted grayscale is given as:

- Red Channel = Green Channel = Blue Channel = 65535 - $p_{val}$

where $p_{val}$ is the image pixel value from 0-65535

2.1.14 Log

Log colormap converts pixel values to a logarithmic scale. In such a scheme, the values represented by colors increases exponentially. It is defined as:

$$p = [(2^{bitdepth} - 1) \times \log_{bitdepth}(p_{val})]$$

where $p_{val}$ is the image pixel value and $p$ is the value in log colormap
2.2 Edge Detection Algorithms

Edge detection [Lin98] [ZT98] is the process of identifying points of sharp brightness changes and discontinuities in an image. The discontinuities can be sudden changes in pixel intensities, depth, surface orientation etc., and describe the object boundaries in a scene.

There are many algorithms for finding edges in an image, but most of them can be grouped into two categories - ‘edge detection methods’ and ‘edge strength methods’. SKIPT has edge strength operators like Sobel Operator, Prewitt Operator, Robert Cross Operator, Laplacian, and Laplacian of Gaussian. Under the ‘edge detection methods’, Canny has been implemented. It is discussed in this section.

2.2.1 Canny

Canny edge detector [Can86] [Gre] is a ‘true edge detector’. Edge detection methods based on first derivative operators or second derivative operators (described in later sections) are essentially edge strength operations as opposed to edge detection operations. Canny on the other hand gives a yes or no decision for the presence of the edge. It has a low error rate, the edge points obtained are well localized, and it gives only one response to a single edge. The algorithm involves the following steps.

Step 1 - Image Smoothing
To reduce the detector’s sensitivity to noise, the subject image is convolved with a Gaussian function and smoothed. Owing to its computation simplicity and effectiveness, Gaussian smoothing is the preferred smoothing technique. The radius of the Gaussian kernel used decides the sensitivity to noise. If the width is too small then its robustness to noise is less. If the width is too large then the subject will get blurry and the localization error will increase. However with the right width, we can attain a balance between minimizing the localization error and also maximizing the sensitivity to noise.

Step 2 - Edge Strength Calculation
The gradient of the image is used to find the edge strength. The Sobel operator is applied at each point in the image, and gives the vertical edges and horizontal edges.
Step 3 - Computing Edge Direction
For every point in the image the edge direction is calculated by taking the inverse tangent of the vertical derivative divided by the horizontal derivative (similar to equation 2.3).

Step 4 - Rounding off the Edge Direction
The edge direction for a pixel can be value between $0^\circ$ and $179^\circ$. For algorithm processing purposes, it needs to be rounded off to a value traceable in an image. Considering a $3 \times 3$ image neighborhood, the pixels are arranged in the following fashion:

\[
\begin{array}{ccc}
 x & x & x \\
 x & p & x \\
 x & x & x \\
\end{array}
\]

From an 8-connected neighbor approach, pixel 'p', has four possible directions - 0 degrees (pixels on the right and left), 45 degrees (pixels on the top right corner and bottom left corner), 90 degrees (pixels on the top and bottom), or 135 degrees (pixels on the top left corner and bottom right corner). For every point, the edge direction has to be mapped into one of these four directions.

Figure 2.6 [Gre] helps in visualizing the idea.

- If the edge direction value is between $0^\circ$ to $22.5^\circ$ or $157.5^\circ$ to $180^\circ$, it is assigned a value of $0^\circ$. (yellow range)
- If the edge direction value is between $22.5^\circ$ to $67.5^\circ$, it is assigned a value of $45^\circ$. (green range)
Figure 2.6 [Gre] Orientation Assignment

range)

- If the edge direction value is between $67.5^0$ to $112.5^0$, it is assigned a value of $90^0$. (blue range)
- If the edge direction value is between $112.5^0$ to $157.5^0$, it is assigned a value of $135^0$. (red range)

Step 5 - Non-Maximum Suppression
Non-maximum suppression is used to keep only those pixels on an edge with the highest gradient magnitude and removes all pixel values that are not part of an edge, i.e. keep edges one pixel wide. This step helps in thinning of the edges and bringing out discernible features.

Step 6 - Hysteresis Thresholding
The last step - hysteresis thresholding is used for eliminating streaking and achieving a balance between false positive edge points and false negative edge points. Because of noise, using a single threshold has its limitations. A high threshold removes significant information, while a low threshold leads to generating numerable false edge points. To avoid this, hysteresis uses 2 thresholds - $(T_{upper})$ and $(T_{lower})$. Not only does this reduces false edge points in the output image, but also links the edges and reduces discontinuities.

1. If a pixel in the image has a value greater than $T_{upper}$, it is accepted to be an edge pixel.
2. If a pixel in the image has a value lower than $T_{lower}$, it is rejected.
3. Pixels that have a value greater than $T_{lower}$, and are connected to edge pixel obtained from (1) are also selected as edge pixels.
2.3 Image Derivatives

As stated earlier, image derivatives helps in identifying regions of discontinuities in an image. They corresponds to a directional change in the intensity or color of image.

2.3.1 First Derivative X Direction, First Derivative Y Direction, Second Derivative X Direction, Second Derivative Y Direction

These functions find the image derivatives by calculating the difference between the front pixel and rear pixel in the respective direction. For instance, the first derivative in y direction at a point $P[column][row]$ will be given by:

$$\frac{P[column][row+1]-P[column][row-1]}{2}$$

Similarly, the first derivative in x direction at a point $P[column][row]$ will be given by:

$$\frac{P[column+1][row]-P[column-1][row]}{2}$$

2.3.2 First Derivative X Direction Gaussian, First Derivative Y Direction Gaussian, Second Derivative X Direction Gaussian, Second Derivative Y Direction Gaussian

These functions find the image derivatives by calculating the respective derivative of an isotropic, zero mean Gaussian function, and applying it to the image.

2.3.3 First Order Derivative Operators Used in Edge Detection

2.3.3.1 Sobel Operator

The Sobel operator is a popular operator used in edge detection algorithms. It is a discrete differentiation operator which uses Gaussian smoothing. The operator calculates the gradient of image intensity at each point. A greater difference in intensity corresponds to a higher magnitude of derivative and thus gives a clearer edge. The operator consists of a
pair of $3 \times 3$ convolution kernels as shown in figure 2.7. One kernel is simply the clockwise rotation of the other.

\[
\frac{1}{8} \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \ast I \\
\frac{1}{8} \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \ast I
\]

(a) Detecting Vertical Edges ($G_X$)  
(b) Detecting Horizontal Edges ($G_Y$)

**Figure 2.7** Edge Detection Using Sobel Operators

Here $I$ is the image to be operated and $\ast$ represents convolution operator.

Each of these two kernels help in calculating the gradient along each of the two perpendicular orientations. $G_X$ can be the horizontal gradient and $G_Y$ can be the vertical gradient. Gradient of an image is a 2-dimensional vector. The gradient along the horizontal direction gives the vertical edges, while the gradient along the vertical direction gives the horizontal edges. Combining the individual gradients together gives the absolute magnitude of the gradient and the orientation of that gradient at each point. The gradient magnitude is found by:

\[
|G| = \sqrt{G_X^2 + G_Y^2} \tag{2.1}
\]

An approximate magnitude is calculated using:

\[
|G| = |G_X| + |G_Y| \tag{2.2}
\]

which is much faster to compute.

The angle of orientation of the edge, relative to the pixel grid is given by:

\[
\theta = \arctan\left(\frac{G_Y}{G_X}\right) \tag{2.3}
\]
2.3.3.2 Prewitt Operator and Robert Cross Operator

Prewitt Operator and Robert Cross Operator are very similar to Sobel Operator. The difference lies in the use of convolution kernels. Both of them are discrete differentiation operators.

The kernels used in the case of Prewitt Operator are shown in figure 2.8. These are among the 8 kernels that can be used, and are obtained by rotating the coefficients of a kernel circularly. Each of the kernel thus obtained is used to detect a particular edge orientation from $0^\circ$ to $315^\circ$ in steps of $45^\circ$, where $0^\circ$ corresponds to a vertical edge.

\[
\begin{bmatrix}
1 & 0 & -1 \\
1 & 0 & -1 \\
1 & 0 & -1 \\
\end{bmatrix} \ast I \quad \frac{1}{6}
\]

(a) Detecting Vertical Edges ($G_X$)

\[
\begin{bmatrix}
1 & 1 & 1 \\
0 & 0 & 0 \\
-1 & -1 & -1 \\
\end{bmatrix} \ast I \quad \frac{1}{6}
\]

(b) Detecting Horizontal Edges ($G_Y$)

Figure 2.8 Edge Detection Using Prewitt Operators

The kernels used in the case of Robert Cross Operator are the following:

\[
\begin{bmatrix}
1 & 0 \\
0 & -1 \\
\end{bmatrix} \ast I \quad \begin{bmatrix}
0 & 1 \\
-1 & 0 \\
\end{bmatrix} \ast I
\]

(a) ($G_X$) (b) ($G_Y$)

Figure 2.9 Edge Detection Using Robert-Cross Operators

The only difference between Prewitt’s and Roberts’ Edge Detector lies on the Gaussian-kernel mask that is used. The Roberts Edge detector is faster since the filter is small but it is also prone to interference by noise. If edges are not very sharp the filter will tend to not detect the edges.
2.3.4 Second Order Derivative Operators Used in Edge Detection

The zero-crossings of a non-linear differential function correspond to edges. Methods which find edges by using zero-crossings of a second order derivative operator, usually a Laplacian are also called as zero-crossing based methods.

2.3.4.1 Laplacian

“The Laplacian is a 2-D isotropic measure of the second spatial derivative of an image” [RFW]. It works by emphasizing those regions, and highlighting pixels which vary greatly in intensity from their immediate neighborhood. Such regions/pixels essentially correspond to the edges.

Deriving edges using the Laplacian method requires the use of only one kernel as it calculates the second order derivatives in a single pass. Some of the commonly used kernels are the following:

\[
\begin{bmatrix}
0 & -1 & 0 \\
-1 & 4 & -1 \\
0 & -1 & 0 \\
\end{bmatrix}
\]

(a) Laplacian Operator

\[
\begin{bmatrix}
-1 & -1 & -1 \\
-1 & 8 & -1 \\
-1 & -1 & -1 \\
\end{bmatrix}
\]

(b) Laplacian Operator (include diagonals)

Figure 2.10 Laplacian Kernels

For better approximation instead of a 3x3 kernel, a 5x5 kernel can be used. It is shown in figure 2.11.

2.3.4.2 Laplacian of Gaussian

If a Laplacian kernel is applied to an image on which a Gaussian smoothing kernel has already been applied, then it is called Laplacian of Gaussian (LoG) [Gun99] [Wan]. The idea is to reduce the sensitivity to noise. A grayscale image is used as input.

In an alternate approach, the LoG $\Delta G(x, y)$ can be computed first and then convolved with the input image. The latter approach has been discussed below.
\[
\begin{bmatrix}
-1 & -1 & -1 & -1 & -1 \\
-1 & -1 & -1 & -1 & -1 \\
-1 & -1 & 24 & -1 & -1 \\
-1 & -1 & -1 & -1 & -1 \\
-1 & -1 & -1 & -1 & -1 
\end{bmatrix}
\]

Figure 2.11 A 5x5 Laplacian Kernel

Consider a Gaussian kernel of width \( \sigma \)

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

Taking first partial derivative with respect to \( x \) we obtain

\[
\frac{\partial}{\partial x} G_\sigma(x, y) = \frac{\partial}{\partial x} e^{-\frac{x^2+y^2}{2\sigma^2}} = -\frac{x}{\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}
\]

and second partial derivative with respect to \( x \)

\[
\frac{\partial^2}{\partial x^2} G_\sigma(x, y) = \frac{x^2}{\sigma^4} e^{-\frac{x^2+y^2}{2\sigma^2}} - \frac{1}{\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} = \frac{x^2 - \sigma^2}{\sigma^4} e^{-\frac{x^2+y^2}{2\sigma^2}}
\]

For simplifying the calculations, the normalizing coefficient \( \frac{1}{\sqrt{2\pi\sigma^2}} \) has been ignored from all the calculated partial derivatives. Similarly, second partial derivative with respect to \( y \) will be

\[
\frac{\partial^2}{\partial y^2} G_\sigma(x, y) = \frac{y^2 - \sigma^2}{\sigma^4} e^{-\frac{x^2+y^2}{2\sigma^2}}
\]

The convolution kernel used to calculate the LoG is the sum of second partial derivatives of \( x \) and \( y \), and is given as:

\[
LoG \overset{\Delta}{=} \Delta G_\sigma(x, y) = \frac{\partial^2}{\partial x^2} G_\sigma(x, y) + \frac{\partial^2}{\partial y^2} G_\sigma(x, y) = \frac{x^2 + y^2 - 2\sigma^2}{\sigma^4} e^{-\frac{x^2+y^2}{2\sigma^2}}
\]

The Gaussian, its first derivative, and its second derivative are shown in figure 2.12 [Wan] and figure 2.13 [Wan].

An LoG kernel obtained by evaluating the LoG expression \( \Delta G_\sigma(x, y) \) can also be used to approximate the same result. The size of the kernel is left up to the user’s discretion.
Figure 2.12 [Wan] Gaussian (red), Gaussian First Derivative (green), and Gaussian Second Derivative (blue)

Figure 2.13 [Wan] Plot of Laplacian of Gaussian
However, since for a homogeneous region the convolution is zero, therefore the sum of all the elements of any LoG kernel have to be zero [Wan].

\[
\begin{bmatrix}
0 & 0 & 1 & 0 & 0 \\
0 & 1 & 2 & 1 & 0 \\
1 & 2 & -16 & 2 & 1 \\
0 & 1 & 2 & 1 & 0 \\
0 & 0 & 1 & 0 & 0
\end{bmatrix}
\]

**Figure 2.14** A 5x5 LoG kernel

### 2.4 Feature Detectors

A feature is a “point of interest”. Technically, “features correspond to a sparse set of local measurements that capture the essence of the underlying input images and that encode their interesting structure” [GL11].

Some of the properties of a good feature include:

- Consistency. It should be repeatable for several images of the same scene.
- Invariant to translation, rotation, and scale.
- Resistance to noise.
- Invariant to occultness. When some parts of a shape are hidden or overlapped by other objects, the feature points of the remaining part must not change.
- Statistically independent.
- Salient and identifiable. The feature point should stand out to human eye.

There are various feature detection methods. They have been divided into four main groups- *Edge Detectors*, *Corner Detectors*, *Blob Detectors*, and *Region Detectors*. Figure 2.15 [BVS10] lists out the important methods under each group.

From this broad classification *Harris, Harris Laplace*, and *SIFT* have been discussed in this section.
2.4.1 The Harris Detector

Harris Detector [HS88] uses a gradient formulation, a second-moment matrix to detect changes in intensity over a window/region, and return keypoints. These keypoints correspond to corner points of an image. Figure 2.16 explains the idea clearly.

At every point in the image x, the second-moment matrix/Harris matrix H is calculated. Note that the matrix H is the same as matrix C shown in figure 2.16. The Harris matrix is multiplied by an isotropic two dimensional Gaussian function \( G_\sigma(x) \). The expression is given as:

\[
H(x,\sigma) = G_\sigma(x) \ast \begin{bmatrix}
I_x^2(x,\sigma) & I_x I_y(x,\sigma) \\
I_x I_y(x,\sigma) & I_y^2(x,\sigma)
\end{bmatrix}
\]

Convolution with the Gaussian \( G_\sigma(x) \) ensures that an entire neighborhood around the test point is considered. It is essentially the window over which the change needs to be observed. By analyzing the magnitude of the eigenvalues (\( \lambda_1 \) and \( \lambda_2 \) of the Harris matrix,
we can find out whether or not a pixel is a point of interest.

- If $\lambda_1$ and $\lambda_2$ are low or nearly zero, then this pixel $(x, y)$ has no features of interest.
- If either $\lambda_1$ is small (nearly equal to zero) and $\lambda_2$ has some large positive value or vice-versa, then an edge point is found.
- If both $\lambda_1$ and $\lambda_2$ have large positive values, then a corner point is found.

If calculation of the eigen values of the Harris matrix is too cumbersome, then the following expression can be used

$$\text{det}(H) - \alpha \text{trace}^2(H) > t,$$

which avoids the need to compute the exact eigenvalues. Typical values for $\alpha$ are in the range of 0.04 - 0.06. The threshold value $t$ is set by the user.

### 2.4.2 The Harris Laplace Detector

“The Harris-Laplace detector uses a scale-adapted Harris function to localize points in space and then selects the points for which the Laplacian of Gaussian attains a maximum over scale” [MS02] [MS04]. This formulation makes the detected points invariant to scale.
The algorithm consists of three steps:
1. Using the Harris Detector to detect points at a predefined scale
2. Using Laplacian to construct a scale space for the output returned by Harris Detector
3. Iteration over the scale space to determine the scale and location of the feature points

After obtaining the feature points returned from Harris Detector, a scale-space representation for pre-selected scales $\sigma_n = \xi^n \sigma_0$ is constructed. Here $\xi$ is the scale factor between successive levels. Every pixel is compared to its 8 neighbors at the same level of the scale-space. If it is the maximum value in this level, the closest pixel location is calculated at the level lower than the current level in the scale-space. If the pixel still remains higher, the test is repeated for the level above the current level in scale-space, considering a total of 26 neighbors of a point $x$. A threshold is used to reject the points having a small value of 'cornerness'.

$$H_L(x, \sigma_I, \sigma_D) = \begin{bmatrix}
H_{L_{11}} & H_{L_{12}} \\
H_{L_{21}} & H_{L_{22}}
\end{bmatrix} = \sigma_D^2 G(\sigma_I) * \begin{bmatrix}
L_{x_I}^2(x, \sigma_D) & L_x L_y(x, \sigma_D) \\
L_x L_y(x, \sigma_D) & L_y^2(x, \sigma_D)
\end{bmatrix}$$

where

$$L_x = I(x) * G_x(x, \sigma_D)$$
$$L_y = I(x) * G_y(x, \sigma_D)$$

The matrix $H_L(x, \sigma_n)$ is computed with the integration scale $\sigma_I = \sigma_n$ and the local scale $\sigma_D = s \sigma_n$, where $s$ is a constant factor usually set to 0.7.

$$\text{Corner} = \det(H_L(x, \sigma_I, \sigma_D)) - \alpha \text{trace}^2(H_L(x, \sigma_I, \sigma_D)) > \text{threshold}$$

$\alpha$ is a constant whose value is selected between 0.04 - 0.06.

### 2.4.3 The SIFT Detector

The Scale-invariant Feature Transform (SIFT) [Low99] detector works by identifying scale-invariant features using a filtering approach. It consists of two major stages:

1. Scale-space generation
2. Keypoint detection

A scale space generated by Difference-of-Gaussian (DoG) is used to identify potential interest points. These points are invariant to scale and orientation. Since DoG is a good approximation to Laplacian of Gaussian (LoG) and has better computation speed, it is used instead of LoG. From the potential key points, the true key points in the scale space are selected by looking for locations that are maxima or minima of a DoG.

\[
D(x, y, \sigma) = (G(x, y, k\sigma) - G(x, y, \sigma)) \ast I(x, y) = L(x, y, k\sigma) - L(x, y, \sigma)
\]

where \( \ast \) is the convolution operator, \( G(x, y, k\sigma) \) is a variable-scale Gaussian, \( I(x, y) \) is the input image, and \( D(x, y, \sigma) \) is Difference of Gaussians with scale \( k \) times larger.

To construct a DoG scale space, the input image is first convolved with a Gaussian function having a \( \sigma \) value of \( \sqrt{2} \). The convolved image is again convolved with the same Gaussian function so that the net value of Gaussian sigma becomes \( \sigma = 2 \). Then the image with \( \sigma = \sqrt{2} \) is subtracted from the image with \( \sigma = 2 \), which leads to a ratio of \( \sqrt{2} \) between the two Gaussians. The next step involves re-sampling of the smoothed image. Using bilinear interpolation, the same procedure is repeated with a pixel spacing of 1.5 in each direction. This process can be better seen in figure 2.17 [Low99].

In the keypoint detection step the low contrast points, and edge points are rejected. Maxima and minima of the scale-space function are determined by the method similar to the one described in Harris Laplace Detector section. A pixel is compared with its 26 neighbors. Figure 2.18 [Low99] visualized the idea.

2.5 Image Segmentation

Image segmentation is the technique of partitioning an image into distinct regions, and grouping similar pixels together. The purpose is to provide a meaningful, useful, and easier representation of image analysis. Segmentation techniques can be divided into following categories [Uni]:

1. **Threshold based segmentation** - A threshold value based either on intensity or color can be used to segment the image.
Figure 2.17 Difference of Gaussians in SIFT [Low99]

Figure 2.18 Keypoints detection in SIFT [Low99]
2. **Edge based segmentation** - The edges detected by appropriate algorithms are assumed to represent object boundaries, and are used for segmentation.

3. **Region based segmentation** - These techniques identify objects by starting a ‘region’ at any arbitrary point inside (outside) of an object and then “growing” (“shrinking”) the region outwards (inwards) until the region meets the object boundaries.

4. **Clustering techniques** - These methods attempt to group together similar patterns.

5. **Matching** - When we have an approximate information about the object we wish to identify in an image, we can use this knowledge to locate that object.

Perfect image segmentation - every pixel being accurately assigned to the rightful object, is usually a difficult task. This is because of the occurrence of *oversegmentation* or *undersegmentation*. In the former, pixels belonging to a single object are treated as belonging to different objects, resulting in a single object being represented by more than one objects. The latter case is the opposite. Pixels belonging to two or more different objects are classified as belonging to a single object. Figure 2.19 explains the idea of *oversegmentation* and *undersegmentation*.

![Figure 2.19](YL)

Figure 2.19 [YL] Illustrating the stages between *oversegmentation* (extreme left) and *undersegmentation* (extreme right)

**Convex Hull, Active Contours/Snakes, Color Based Segmentation, and Watershed Segmentation** have been implemented in SKIPT.

### 2.5.1 Convex Hull

The *Convex Hull* [CSA] for some random points P in a plane is the smallest convex set which contains P. A convex polygon is unique and its vertices are points from P.
Take a piece of wood with a number of metal pins hammered into it randomly (as shown in figure 2.20 [Ala]). Bring an elastic rubber band, stretch it sufficiently wide so that all metal pins are inside it, and let it go. We will observe that the elastic rubber band shrinks and encloses around only the outer metal pins. These outer metal pins form the vertices of convex hull, and the shape of the elastic band is the convex hull.

![Convex Hull Diagram](image-url)

Figure 2.20 [Ala] The convex hull ideology

There are several algorithms to solve the convex hull problem with varying runtimes. The one implemented in SKIPT uses the popular Graham’s scan algorithm, having a complexity of $O(n \log n)$, where $n$ is the number of points. The Graham’s scan algorithm starts with a point that is on the convex hull for sure and then adds points to the convex hull iteratively.

1. Let $H$ be the list of points which forms the convex hull. It is empty initially.
2. Choose a point $p_0$ which has the lowest $y$-coordinate and add it to $H$.
3. Sort the remaining points $(p_1, p_2, \ldots, p_n)$ in ascending order by their polar angles relative to $p_0$.
4. For each point $p_i$, consider the last two points ($p_{i-1}$ and $p_{i-2}$) from the points added in $H$:
   (a) If $p_i, p_{i-1}$ and $p_{i-2}$ make a “left turn”, add $p_i$ to $H$
   (b) If $p_i, p_{i-1}$ and $p_{i-2}$ make a “right turn”, keep removing points $(p_{i-1}, p_{i-2}, \ldots)$ from $H$ until the set $p_i, p_{i-1}$ and $p_{i-2}$ makes a left turn
2.5.2 Active Contours/Snakes

An active contour or snake [KWT88] [Uni] is an energy minimizing spline, a user defined curve in an image that is allowed to change its location and shape until it best satisfies predefined conditions. Since the snake has the inherent nature of forming an enclosure around the object boundary, it can be used in segmentation.

The modeling of a snake \( C \) is carried out as a parametrized curve

\[
C(s) = (x(s), y(s)) \quad 0 \leq s \leq 1
\]

\( C(0) \) gives the starting point coordinates, while \( C(1) \) gives the end point coordinates. The movement of the snake is an energy minimization process, and the total energy \( E \) to be minimized consists of three terms:

\[
E = \int_{0}^{1} E(C(s)) \, ds = \int_{0}^{1} \left( E_i(C(s)) + E_e(C(s)) + E_c(C(s)) \right) \, ds
\]

The term \( E_i \) is based on internal forces of the snake - it is a function of the size and shape of the snake, and independent of the image being segmented. Whenever the snake is stretched or bent, \( E_i \) increases. The term \( E_e \) is based on external forces. The farther the snake is from the desired part of the image, the more is the external energy. Finally, \( E_c \) can be used to impose additional constraints/penalties on snake to make it more robust. This can avoid creation of loops in the snake, prevent the snake from going into undesired image region, etc. For many applications, \( E_c \) is not used and set to zero. The internal energy and external energy are mathematically represented as

\[
E_i = c_1 \left\| \frac{dC(s)}{ds} \right\|^2 + c_2 \left\| \frac{d^2C(s)}{ds^2} \right\|^2
\]

\[
E_e = -c_3 \left\| \nabla f \right\|^2
\]

Here the external term is modeled for making the snake attract to edges of a given image \( f \). The external term can be changed to make the snake follow ridges, find corner points, among others. The influence each term has on the movement of the snake is determined by the constants \( c_1, c_2, \) and \( c_3 \). For instance, \( c_1 \) controls the elasticity of the snake, and setting \( c_1 \) to zero will result the snake to stretch infinitely. Even though the snake will be stretching,
there will be no change in the energy as $c_1$ is zero. Conversely, if the value of $c_1$ is too high, then the snake will have very low elasticity. It won't be able to stretch much. Similarly $c_2$ controls the stiffness of the snake. Overall, the values of $c_1$, $c_2$, and $c_3$ influence the behavior greatly and their values should be set carefully.

**Figure 2.21** [BMS97] Example of a snake. The sequence of images shows the evolution of the snake from a user-defined original position to its final state

### 2.5.3 Color Based Segmentation

Segmentation based on partitioning of the color space is known as ‘color based segmentation’ [Webd]. The user selects a dominant/reference color $(R_0, G_0, B_0)$ which he/she wants to extract from the image $(f(x,y))$, and from every pixel color in image $(R(x,y), G(x,y), B(x,y))$, a Cartesian distance is taken from the reference color. The resultant value is then thresholded.

$$
g(x, y) = \begin{cases} 
1 & d(x, y) \leq T_{val} \\
0 & d(x, y) > T_{val}
\end{cases}
$$

$$
d(x, y) = \sqrt{(R(x, y) - R_0)^2 + (G(x, y) - G_0)^2 + (B(x, y) - B_0)^2}
$$

Here $g(x, y)$ is the value of the pixels of the resultant binary image. Based on the thresholding value, regions in the input test image bearing similar tones as that of the reference color are highlighted in black (or white) in the output binary image. Another interpretation of the thresholding rule is simply a definition of a sphere whose center is the reference color, and the threshold value is the radius. Regions inside the sphere are assigned a high value (value 1), while the regions outside the sphere are assigned a low value (value 0).
2.5.4 Watershed Segmentation

Consider a grayscale image to be a topographic surface. In this context, the higher grayscale intensities correspond to the ‘altitude’/‘peak’, while the lower grayscale intensities denote ‘valleys’ and ‘shallow’ areas. We start flooding a valley with water up to the height at which there is no merging/flow of water from neighboring valleys. The place where the waterfronts meet are called as ‘dams’. Filling of valleys with water continues until all the peaks are submerged and there is no overflow of water from other regions. The idea is to have water stored in a compartmentalized fashion. Finally, when the image is fully flooded, all dams together form the watershed [BM11] of an image. The watershed of an ‘edgeness image’ can be used for segmentation. Figure 2.23 shows how this works. The watershed correspond to objects, it shows object boundaries.

Figure 2.22 [ZMLW09] The concept of watershed

To obtain a good segmentation, it is a usual practice to perform some pre-processing and post-processing techniques on the watershed image. Filling up of the boundaries is a typical post-processing step to obtain solid segments.
2.6 Miscellaneous Operations

2.6.1 Line Profile

A line profile [Cor] [Nur] is a graph showing the variation of pixel intensities along a line for an image. It is a useful tool which can give valuable analysis for various object boundaries in a scene, help identify patterns, and tell user about the color composition.

The shape of the graph and the magnitude of intensity values illustrates the level of contrast between the subjects under consideration. If there is a sudden rise in the intensity then the peak of the graph is steep, denoting a high contrast between the object and its background. For homogeneous regions and areas of constant intensities, the graph is flat. Lastly, if there is a lot of noise in the image then we get a series of narrow peaks. Figure 2.24 [C.T] illustrates this theory. The red line indicates where the line selection was made. Note that the line was drawn from top to bottom.
2.6.2 Image Histogram and associated Image Statistics

A histogram [Col] is a graph of the total number of pixels at each intensity/color level. It shows the tonal range (brightness value) of an image. Analyzing the histogram data can provide useful insight in attaining better quality images by adjusting certain image conditions [Cor]. Two such parameters are described below.

- **Underexposure and Saturation** - If the lighting conditions in which the images are taken are low then the images are underexposed. For such an image, the histogram
has a higher concentration of low gray values pixels. On the other hand, if the lighting conditions in which the images are taken are high then the images are saturated. The histogram for such images has a higher concentration of high gray values pixels. It is important to compensate for such conditions so that better images can be obtained.

- **Contrast** - Contrast is the variation between light areas and dark areas in a scene. The width of an histogram reflects the amount of contrast in an image. If the width is large and the histogram has sufficient pixels at lower color levels as well as higher color levels, then it indicates a greater difference between light regions and dark regions and consequently a higher contrast. Conversely, if the width is small then it indicates a small difference between light regions and dark regions and a lower contrast. For instance pictures taken under cloudy conditions or less light have low contrast and a narrow histogram. They appear dull. On the other hand, pictures taken on a sunny afternoon have significantly higher contrast and a broad histogram. They appear bright and vivid.

It is also possible to segment objects based on their intensities from the information obtained by the histogram.

In addition, SKIPT provides fundamental image information such as the mean, maximum pixel value, minimum pixel value, and standard deviation. This helps in getting a better understanding of the image analysis.

### 2.6.3 Acquiring pixel intensities from user selected points

SKIPT comes with the functionality of returning pixel intensities from any user selected point on both the images - original and current.
Chapter 3

Comparison With Similar Existing Software

This chapter describes the existing applications that are somewhat similar to SKIPT. Although the functionality might be related in some aspects, the development objectives and the served user base is different. The similarities and differences of such software packages have also been discussed.

3.1 MATLAB/Simulink

MATLAB (Matrix Laboratory) is a fourth-generation programming language which provides an interactive environment for numerical computation, iterative exploration, visualization, design, and problem solving. It is a platform having development tools for improving maintainability of the code and the quality of the code modules. The rich Image Processing and Computer Vision toolbox with numerable built in algorithms makes it an ideal tool for implementing and testing image processing and analysis algorithms. However, MATLAB is more of a programming environment requiring the user to manually write the code to test its performance, rather than providing the users with customizable codes and allowing them to evaluate the performance merely by changing the concerned parameters.

Simulink on the other hand is a block diagram based simulation environment using a Model-Based Design approach. Equipped with a graphical editor, block libraries which are customizable, and solvers for modeling and simulating dynamic systems, it is a powerful
software. Although the Computer Vision System toolbox does have some pre-processing image processing algorithms, but the supported image data-types are extremely limited. Having a confined set of operations restricts its usability. In addition there is no access to the source code of the employed algorithms. It is more of an image pre-processing tool rather than a learning tool.

3.2 Wolfram Mathematica

Mathematica is a symbolic mathematics software package. It is a ‘high level complete system’ and is capable of performing all the numeric mathematics and matrix operations that MATLAB does. Much of the wider functionality for MATLAB is available as additional toolboxes - MATLAB has toolboxes for curve fitting, statistics, optimization, symbolic math, Image Processing etc., but all these functionalities are built in to Mathematica already. MATLAB’s language has a procedural development style while Mathematica allows programming in procedural, functional, object oriented and rule based styles.

In summary, there is a big overlap between Mathematica and MATLAB. The block diagram environment of Mathematica however is not as comprehensive as Simulink.

3.3 ImageNets

ImageNets [Webe] is an image processing block diagram based programming environment which uses OpenCV libraries. The motivation behind its development is rapid prototyping of Robot Vision algorithms in a user-friendly style. It also has a GUI designer which eliminates the need of learning any programming language to perform operations on image. The objectives for ImageNets' development are:

1. Visualization of Robot Vision results in 3D
2. Rapid prototyping of Computer Vision programs using OpenCV’s faster execution speeds
3. Realizing Computer Vision feedback loops

An ImageNet is created using ImageNet Designer and saved as an XML-File. Using the designer, the algorithms are fine-tuned and the results are visualized in 2D and 3D.
ImageNets uses OpenCV libraries and has a lot of external dependencies. There is no support for complex images or raw image formats. Although there are about 300 functional blocks implemented in ImageNets, most of them are unstable (according to ImageNets website) and the functional blocks’ robustness has not been tested.

### 3.4 Computer Vision and Image Processing Tools

Computer Vision and Image Processing Tools (CVIPtools) [DEK12][Tea][Umb10] is an Open Source image processing software. It is a collection of computer imaging tools providing services to the users at four layers - the algorithms code layer (C function layer), the Common Object Module (COM) interface layer, the CVIP Image layer and the Graphical User Interface (GUI). The C function layer consists of all image processing functions. The Common Object Module (COM) interface layer’s purpose is to link together the C function layer and the GUI. The CVIP Image layer provides an Object Oriented Programming (OOP) approach for the C function layer. Finally, the GUI implements the image queue and manages the input from the user and resultant output of the concerned operation.

CVIPtools provides support for reading common image formats including TIFF, PNG, GIF, JPEG, and BMP. Basic image processing functions such as edge detection, feature
detection, morphological filters, spatial domain image restoration, logical and arithmetical operations between images, image smoothing and sharpening etc., are all part of the software package.

The drawbacks of CVIPtools are that there is no support for complex images, raw image formats, and lack of block diagram environment.

### 3.5 GNU Image Manipulation Program

The GNU Image Manipulation Program (GIMP) is a versatile graphics manipulation tool. It can be used for numerable image manipulation tasks. GIMP is very expandable and extensible with support for plugins, making it easier to script complex image manipulation operations. It is an excellent image pre-processing tool providing comprehensive filters - blur filters, enhance filters, distort filters, noise filters, edge-detection filters, and artistic filters, among other features.

Overall, GIMP's target audience constitute of artists looking for a reliable open source substitute to Photoshop rather than the scientists looking for programs for evaluating algorithm performances. This makes GIMP a suitable software for artistic manipulation and picture enhancement.

### 3.6 Other Image Processing/Image Viewing Software Packages

Softwares like Adobe Photoshop, digiKam, iPhoto, IrfanView, Picasa, Zoner Photo Studio etc. are popular image processing programs with goals of primarily being an image viewing, image enhancement and artistic manipulation software. SKIPT uses some of the functions supported by these packages.
Table 3.1 Comparison of various softwares

<table>
<thead>
<tr>
<th></th>
<th>SKIPT</th>
<th>MATLAB/Simulink</th>
<th>Mathematica</th>
<th>ImageNets</th>
<th>CVITools</th>
<th>GIMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supported Image Formats (Read)</td>
<td>BMP,GIF, JPEG, PNG,PBM, PPM, TIFF, XBM, XPM,IFS, etc.</td>
<td>BMP,GIF, JPEG, PNG,PBM, PPM, TIFF, XWD, etc.</td>
<td>BMP,GIF, JPEG, PNG, PBM, PPM, PGM, TIFF, etc.</td>
<td>BMP,GIF, JPEG, PNG, PBM, PGM, PPM, TIFF, etc.</td>
<td>BMP,GIF, JPEG, PNG, PGM, PPM, TIFF, etc.</td>
<td>BMP, JPEG, PNG, TIFF, etc.</td>
</tr>
<tr>
<td>Supported Image Formats (Write)</td>
<td>BMP,GIF, JPEG, PNG,PBM, PPM, XBM,IFS, etc.</td>
<td>GIF, HDF4, JPG,PNG, PPM, RAS, TIFF</td>
<td>BMP,GIF, JPEG, TIFF, PNG, PBM, PGM, PPM</td>
<td>BMP,JPG, JPEG, PNG, PPM, XBM, etc.</td>
<td>NA</td>
<td>GIF, JPEG, PNG, TIFF</td>
</tr>
<tr>
<td>Supported Data Types</td>
<td>ALL (complex, double, int, float)</td>
<td>double, uint8, unit16</td>
<td>uint8, uint16, double32, double64</td>
<td>No support for complex formats</td>
<td>No support for complex formats</td>
<td>float, int8, int16, int32, float</td>
</tr>
<tr>
<td>Platforms Supported</td>
<td>Windows, Mac, Linux, Embedded Linux</td>
<td>Windows, Mac, Linux</td>
<td>Windows, Mac, Linux</td>
<td>Windows, Linux</td>
<td>Windows, Linux, Solaris</td>
<td>Windows, Mac, Linux, Solaris, FreeBSD</td>
</tr>
<tr>
<td>Image Dimensions Supported</td>
<td>Infinite</td>
<td>Infinite</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Block Diagram Environment</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes (but not for image processing)</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
Chapter 4

Operational Overview

The objective of this chapter is to give a basic understanding about the working of SKIPT. It has been divided into 2 sections - High Level Operation and Low Level Operation. The contents under ‘High Level Operation’ are more of a “user's guide”, giving the reader a brief understanding of how to operate the software and utilize its functionality. On the other hand, the ‘Low Level Operation’ explains the overall architecture of the application, a bird’s eye view of the coding details, and the building blocks of the application. It is more of a “programmer's guide”.

4.1 High Level Operation

To better understand the user level working of the application, a brief description is provided followed by the appropriate screen-shots.

Every operation begins with the loading of an image into the SKIPT environment. The user clicks the File menu and selects Open. A window appears which lets the user select an image file and loads it in the working space of SKIPT.

Once the image is loaded, the user gets the option to operate on the original image or the current image. The original image remains unchanged throughout the program use, while the current image (on the right) is updated as per the operation implemented by the user. The software gives the functionality to save or print the current image any time, as required.
Now the actual image processing starts. As stated earlier, the various algorithms into are divided 6 major categories according to type and nature (Image Pre-processing, Segmentation, Edge Detection, Feature Detection, Derivatives, and Misc. Operations), making it easy to find the operation needed. Each of these categories has a dedicated menu, listing the supported algorithms (as discussed in Chapter 2).

We begin with **Image Pre-Processing** section. Most of the operations in this category do not have any user defined parameters. *Image Smoothing, Sharpening, various colormaps, and Histogram Equalization* do not take any input from the user. On the other hand, *Brightness, Warmness, and Coolness* operations comes with a slider, which is used to vary the brightness/warmness/coolness values respectively. Figure 4.3 shows the Brightness opera-
tion in action.

![Figure 4.3 Brightness operation in SKIPT. The user with the help of the slider can vary of amount of brightness/darkness desired in the image.](image)

**Figure 4.3** Brightness operation in SKIPT. The user with the help of the slider can vary of amount of brightness/darkness desired in the image.

*Window and Level* operation comes with two sliders - one for setting the upper threshold value and the other for setting the lower threshold value. As mentioned earlier, pixel values darker than the lower threshold value are mapped to black (value 0), while pixels brighter than the upper threshold value are mapped to white (value 255). All pixel values in between the two thresholds are scaled linearly in 0-255 range.

![Figure 4.4 Window and Level Operation](image)

**Figure 4.4** Window and Level Operation
Moving on, in the **Derivatives** section *First Derivative X Direction, First Derivative Y Direction, Second Derivative X Direction, and Second Derivative Y Direction* are all non-user-input operations, while their Gaussian counterparts come with a slider to vary the sigma value.

**Figure 4.5** First Derivative X Direction Operation

![Figure 4.5 First Derivative X Direction Operation](image)

**Figure 4.6** First Derivative Y Direction Gaussian Operation. The slider allows the user to vary the value of sigma used for calculating the derivative.

![Figure 4.6 First Derivative Y Direction Gaussian Operation](image)
For the **Edge Detectors** section, *Sobel, Robert-Cross, Prewitt, Laplace, and Laplacian of Gaussian* come with a slider to set the threshold value. *Canny* on the other hand comes with two sliders - one for upper threshold and one for lower threshold.

![Figure 4.7 Edge Detection using Sobel Operator](image)

**Figure 4.7** Edge Detection using Sobel Operator

![Figure 4.8 Canny Edge Detection Operation](image)

**Figure 4.8** Canny Edge Detection Operation
Similarly, in the **Feature Detectors** section, *Harris* operations uses two user defined inputs - sigma and threshold value for Harris points, while *Harris Laplace* uses three user-defined inputs - sigma, threshold value for Harris points, and threshold value for Laplacian scale space.

In the **Image Segmentation** section, *Color Thresholding* uses four user defined inputs - one for each of the three color bands, and one for the overall thresholding value. *Watershed Segmentation* has no user defined parameters.

Lastly, in the **Misc. Operations** *Image Histogram* does not have any user defined inputs and *Image Profile* requires the user to drag a line across the region where the image profile has to be mapped.

The “show stopper” of SKIPT is the ability to switch to a block diagram based image processing, algorithm simulation, and evaluation environment with a press of a button. The user simply clicks on **Switch to Block Diagram Mode** from the *File* menu and is transferred to an interactive model-based design environment. Figure 4.9 shows the block diagram environment. In this environment all the operations are represented by ‘functional blocks’. A model is created by logically linking blocks together. It can be thought of as an executable file that the user can continuously refine throughout the development process. Each block has associated parameters which the user can adjust accordingly (future work), satisfying the exploratory as well as the performance aspect of the concerned operation. When the user completes a block design, the system also generates an actual code. Thus, merely through block diagrams and without the need of writing extensive codes, the system provides an active analysis and implementation of various image processing/computer vision algorithms.

Every model starts by placing the **Open** (to load the image which gets displayed on a side window), followed by any number of blocks. The blocks can be moved anywhere on the canvas. Once the blocks have been placed, they are “double clicked”. This is a way of letting the system know that the concerned operation has to be placed on an *operations queue*. The **Execute** is placed last, marking the end of every model. Finally, the Execute button is double clicked, and depending upon the blocks used to form the *operations queue*, a program is generated and executed. The output of the whole operation is displayed on a second side window. The first window (containing the original image) is also visible alongside the second window for analysis purposes.
4.2 Low Level Operation

Building of any Qt application requires three fundamental components:

- **QWidget**: A visual element in a user interface e.g. button, label, slider etc.
- **Signal**: Indicators denoting the occurrence of a user action (event)
- **Slot**: A function which is called in response to the emitted signal

The signal and slot mechanism is a unique feature in Qt programming. It helps in establishing connection between objects without the need of objects knowing anything about each other. A `connect` statement is used to achieve this functionality.

```
connect(sender, SIGNAL(signal), receiver, SLOT(slot));
```

Here `sender` and `receiver` are pointers to QObjects (base class of all Qt objects), `signal` and `slot` are function prototypes, and `SIGNAL()` and `SLOT()` are macros which convert their
arguments to a string. The following example gives a better understanding of the concept.

```cpp
class MyWidget : public QWidget
{
    public:
        void setValue(int value);
    private:
        int m_value;
};
```

For the first connect statement note that `slider_laplace_th` is a QWidget of the type QSlider class. It is the sender in this case and emits the signal `valueChanged(int)` with an int argument. This signal is emitted whenever the value of `slider_laplace_th` is changed. The changed value leads to the calling of `setNum(int)` slot with an int argument. Finally, `label_laplace_th` (receiver) displays the changed value.

For the second connect statement, the same sender with the same signal interacts with a different slot. Using ‘this’ as the receiver ensures that the scope of the emitted signal is the entire class. In other words, all the QWidgets defined inside the class are eligible to receive this signal. ‘ValueOnSliderMove_laplace_th(int)’ is a function defined in this class. Upon the change of value of the slider, the associated statements defined inside ‘ValueOnSliderMove_laplace_th(int)’ gets executed.

There is no restriction in terms of the number of connections between signal and slots. A signal can be connected to many slots, many signals can be connected to the same slot, and a signal can also be connected to another signal.

Consider the code from the main.cpp of the SKIPT project. This is the portion responsible for the overall execution of the application.

```cpp
#include <QApplication>
#include "skipt.h"

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    SKIPT *handler = new SKIPT(0);
    handler->show();
}
```
return app.exec();
}

Line 1 includes the definition of QApplication class which manages the GUI application's control flow and main setting. Line 2 includes the main header file. This header file contains the definition for the SKIPT class, list of QAction objects, prototypes of the methods defined inside the class, prototypes for signals, slots, and the initialization along with the type of all the QWidgets to be used within the class, among others. Line 6 creates a QApplication object to manage the resources across the entire application. Since Qt comes with several command-line arguments of its own, argc and argv are required by the QApplication constructor. Line 7 creates a new instance of the type SKIPT which is referenced by the variable named ‘handler’. Line 8 makes the application visible. An application is created hidden so that the user customization can be done before showing it. This avoids flicker and discontinuities in the application. Line 9 passes control of the application on to Qt where the program enters an event loop - a kind of stand-by mode where the program waits for user actions such as mouse clicks and key presses. Events are generated by user actions, and the program executes the concerned functions as response. GUI applications have the element of human interaction. Unlike typical batch programs which process inputs, generate outputs, and terminate without human intervention, GUI applications are interactive. The end result and program termination is dependent on how the user communicates with the system.

Finally, there is a project file (.pro extension) which contains the all the information required by qmake (qmake generates Makefile) to build the application. The source files and header files to be included while developing the Qt application are listed here. When the user starts a new Qt application, the Integrated Development Environment (IDE) creates the default project file(.pro), source files(.cpp), and the header files(.h), which can be edited as needed.

SKIPT uses a variety of QWidgets, inbuilt as well as user defined signals and slots, and numerous features of the Qt framework. The source code is heavily commented and thoroughly explains the functionality.
User inputs an image

If the input image is not an IFS image, it gets converted into an IFS image called IFSglobalwork. Another copy of IFSglobalwork is also created called IFSglobalwork_updated. Both these images have a global scope.

An image of the type QImage is created from IFSglobalwork. This image is only meant for display purposes and projecting output in the Qt environment.

User selects an operation from the menu bar

Depending upon the ‘Signal’ emitted, the concerned ‘Slot’ and associated algorithms are processed.

If the user has selected ‘Operate on Original’ from the image selection dock window, then a local copy of IFSglobalwork serves as the input to the called subroutine. IFSglobalwork_updated is set to the output image of the subroutine.

If the user has selected ‘Operate on Current’ from the image selection dock window, then a local copy of IFSglobalwork_updated serves as the input to the called subroutine. IFSglobalwork_updated is set to the output image of the subroutine.

An image of the type QImage is created from IFSglobalwork_updated. This image is used for projecting output in the Qt environment.

Figure 4.10 Sequence of steps followed during an image operation in SKIPT
Chapter 5

System Design and Analysis

5.1 Why prefer Qt over other IDEs with GUI builders?

SKIPT has been designed and developed using Qt. The motivation for using Qt is driven by the fact that it is a cross-platform application framework with a rich graphical user interface (GUI). It inherits classes for sliders, buttons, labels, boxes, dock windows, message box etc., which are the fundamental building blocks in any GUI environment. In addition, Qt provides a large set of libraries that serve nearly every possible programming purpose. Even for non-GUI features like XML parsing, threads, networking, storage, containers etc., there is a class in Qt. Everything follows a consistent style and is multi-platform. This removes the need to use other libraries, thereby removing any dependencies. Lastly, excellent documentation with useful examples, good technical support, and easy GUI makes Qt an obvious IDE to use.

As stated earlier, the Signal and Slot mechanism in SKIPT is unique, and allows for interaction between the objects without the need of the objects to know anything about each other. The toolkits before Qt used callbacks to achieve this functionality. A ‘callback’ is a pointer to a function. For event notification by the processing function, a callback is passed as an argument to the processing function. This callback is executed (‘called back’) by the processing function as required. Communicating using callback can have certain disadvantages. Since callbacks are not type-safe, it is never a surety that the callback with correct arguments will be called by the processing function. Another disadvantage lies in the strongly coupled nature of callbacks and processing function. The processing function
explicitly requires the processing function to know which callback to call. On the other hand, Qt through its Signal and Slot mechanism, allows the user to connect any signal to any slot even if no parameter matches. Although it results in a runtime warning that a connection between both is not possible, the approach is better than callback which results in system crash for the same scenario.

Finally, one of the chief objectives propelling the development of SKIPT was creating an application having minimum dependencies on external libraries, and using as few libraries as possible. Qt satisfies these specifications. Merely by using Qt and IFS (Image File System) libraries, it has been possible to realize a simple yet a comprehensive and powerful image processing toolkit.

5.2 System Performance

The running time of most algorithms is $O(wh)$ where ‘$w$’ is the width and ‘$h$’ is the height of input image. By using image pointers instead of physically acquiring every pixel row by row and column by column, the running time can be improved.

To ensure that the SKIPT meets the standards of a quality software, testing of the system was done extensively, covering a variety of image formats (IFS, JPEG, PNG, GIF etc), various image data types (unsigned 8 bit, signed 16 bit, 64 bit double etc.), different number of image dimensions (2 dimension, 3 dimension color, multi dimension/multi frame), color images, grayscale images, binary images and numerable algorithm testing methods. While the system performs sufficiently fast for smaller and regular images, for bigger images having size greater than $1600 \times 1200$ (width $\times$ height) the system shows some lag for certain algorithms in displaying output. This can be improved by slightly trading off the real time nature of system. In the current implementation, as the user drags the slider, the output is updated simultaneously. If the system is changed as the output being displayed only after the value has been set (value on slider change) rather than value being constantly changed (value on slider move), the problem is solved.

Considering the implemented algorithms and the system as a whole, following testing methodologies were carried out in order to ensure the maximum robustness of the system.

1. Unit Testing

This included testing of individual units of codes, isolating the smallest piece of
testable software, and determining the correctness of its behavior. All the algorithms were unit tested separately at the lowest level to make them error free and design adherent.

2. **Integration Testing**

Once two or more units were tested individually, they were grouped together and tested as a single unit. The objective here was to ensure that different modules working correctly at an individual level can work correctly when combined together.

Consider for instance the integration testing of ‘brightness’ and ‘coolness’ blocks. Whether the user first brightens an image and then increases its coolness, or the user cools the image first and then increases its brightness; the output should be same (provided the values used for changing brightness and coolness are same in both the cases). Figure 5.1 and Figure 5.2 demonstrates the test.

![Figure 5.1 Brightening an image first and increasing its coolness later](image)

(a) Increasing image brightness by value 50

(b) Increasing the coolness by value 29 of the brightened image obtained from (a)
3. **End-to-End Testing**
   Performed after Unit Testing and Integration Testing, this technique is used to test the system as a whole, determine that all the components are meeting the expected functionality, and ensuring that work-flow of the system is correct. Emphasis is on determining whether the flow of the application from starting point to end point is meeting the desired results or not.

4. **Regression Testing**
   After testing the system as a whole, it was tested to determine that addition of new features/functions to the old system maintains the integrity and correctness of the new system. Such testing makes the software friendly for future expansion and makes the addition any new functionality to the existing system easier.

5. **Stress Testing**
   Stress Testing deals with determining the system's performance, availability, and reliability when the system is made to operate outside the limits of normal operation.
Examples include observing the behavior of system if the user changes the value of slider, tries to perform operations without loading an image, passing parameters outside the computational range etc. Such scenarios have been accounted for in SKIPT, and the system either generates appropriate warning messages through ‘message box’ prompting the user to take the desired action, or ignores certain user actions. The objective is to prevent the crash of the system even in unfavorable conditions.

6. **Load Testing**

Lastly, Load Testing was performed to determine the maximum operating parameters (input, output, and intermediate data) in terms of value, number, and type which the system can withstand and operate with.
Chapter 6

Future Work and Conclusions

6.1 Future Work

Most of the supported algorithms mentioned in Chapter 2 have been implemented in the regular version of SKIPT. Some of the algorithms like SIFT from the Feature Detection section, and Snakes and Convex Hull from Segmentation section have not been implemented. They can be added in the later versions. There is ample scope of adding new operations in both the domains - image processing and algorithm evaluation. New operation categories can be created and/or more operations can be added under existing categories. However, majority of the future work lies in improving the functionality of the block diagram version. The current block diagram version simply demonstrates the “proof of concept” and lays down a basic architecture. Only selected functional blocks, which do not require any user-defined parameters have been implemented and the rest is left as future work. The proposed future system could look like something shown in Figure 6.1.

The Program Editor is a canvas where different image operation blocks are placed. This is the place where the system is defined. From the File menu, the user selects New > Model. Then an operation block is dragged from the tools library and placed on the canvas. The block can be moved anywhere on the canvas and resized as per the user’s convenience. Thus, a block diagram is built by arranging the blocks logically. Every model starts by placing the ‘Image Open Block’ from Program Flow Control (to load the image which gets displayed on a side window), followed by any number of blocks from the Tool Library. The ‘Execute Block’ from Program Flow Control is placed last, marking the end of every model. Finally,
the Execute button is clicked, and depending upon the blocks used, a program is generated and executed. The output of the whole operation is displayed on a second side window. The first window (containing the original image) is also visible alongside the second window for analysis purposes. At user's discretion, the generated code can be viewed and imported to a text file by clicking the 'generate code' button in the Program Toolbar.

**Tools Library** lists the different operations (in the form of blocks) that can be done on the images. The inventory consists of operations divided into 7 major categories according to type and nature (Pre-Processing, Segmentation, Edge Detection, Feature Detection, Derivatives, Arithmetic Operations, and Program Flow Control), making it easy to find the block needed. These form the components to define the system. Every block comes with three specifications- parameters, states, and signals. Parameters are the values that can be changed by the user, States are the image associated variables that change as the function associated with the block executes, and Signals are the inputs that trigger a block as well as the outputs that are generated after the execution of a block.

**The Program Toolbar** lists the basic tools to work with the program environment. Operations like zoom in, zoom out, undo, redo, save, select, pan, and execute are included in
this toolbar. The menu bar contains options to create new canvas window, load existing canvas windows, import, and export models etc.

Output Window displays essential messages, error logs, and compilation status during program execution. It helps the user diagnose the system in case of errors due to incorrect block usage, out-of-bounds parameter values, compile errors etc.

Figures 6.2 - 6.8 describe a proposed structure of the Tools Library, and lists the operations under each category.

It is upto the designer's discretion to decide the number of user input parameters, and the number of inputs/outputs for an operational block. For instance, a Canny Edge Detection block can have 3 parameters - one slider/textbox for setting the lower threshold value, another slider/textbox for setting the upper threshold value, and a third slider/textbox to set the value for Gaussian sigma. Considering the number of data channels, all blocks can be single input single output blocks except the blocks under Arithmetic Operations which usually can be single input double output. Needless to mention, more operations under each category or newer categories can also be added.
Figure 6.3 Edge Detection Operations

Figure 6.4 Feature Detection Operations
Figure 6.5 Derivatives Operations

Figure 6.6 Arithmetic Operations
Figure 6.7 Program Flow Operations

Figure 6.8 Segmentation Operations
6.2 Conclusions

The project meets its objective of developing an user-friendly, cross-platform, and educational image processing toolkit using only Qt and IFS libraries. It is successful in demonstrating a prototype software covering a handful of basic algorithms, and serving the foundation of a balanced, interactive, and simple image software. It is to pave way for more comprehensive programs which expands the two domains - algorithm evaluation and image manipulation, equally.
BIBLIOGRAPHY


APPENDIX
Appendix A

SKIPT User Guide

SKIPT (Snyder Khanna Image Processing Toolkit) is a simple, easy to use, interactive, and cross platform image processing and algorithm evaluation program with the motive to serve as a valuable teaching tool in understanding the fundamental computer vision principles/algorithms. From raw images to popular formats - jpg, jpeg, png, xpm, it supports images with pixels of any data type (complex, double, float, int) and any number of dimensions. SKIPT is a basic yet powerful software intended for students and researchers wishing to delve deeper in image processing/object recognition field. Through its ability to compare the performance measures and working of algorithms by simply varying parameters with a GUI and without the need to actually write the code, it becomes an effective learning medium. The idea is to give users a visual feel, comparison, and an active understanding of essential inter-class and intra-class computer vision algorithms.

A.1 Features

The software is written in C and C++ and developed on the Qt application framework. Qt is a popular cross-platform developing environment for developing GUI application software. Some of the features of SKIPT are:

- Platform independence and native look on every supported platform
- Ability to switch to a block diagram based image processing, algorithm simulation,
and evaluation environment

- Code generation and export
- Ability to switch operations on original image and current images
- Support of many image formats (Read/Write - IFS, BMP, JPG, JPEG, PNG, PPM, XBM, XPM; Read - GIF, PBM, PGM)
- Support of images with pixels of any data type - unsigned 8 bit, signed 8 bit, unsigned 16 bit, signed 16 bit, 32 bit integer, 32 bit floating point, 64 bit double, complex short, complex double, complex float
- Popular segmentation algorithms evaluation - Active Contours (Snakes)*, Convex Hull*, Watershed, Color Thresholding
- Popular edge detection techniques along with comparison - Sobel, Prewitt, Robert Cross, Canny, Laplacian, Laplacian of Gaussian
- Popular feature detection algorithms evaluation along with comparison - Harris, Harris Laplace, SIFT*
- Line Profile - both grayscale and colored images
- Image Histogram with statistics - mean, standard deviation, max pixel value, min pixel value
- Image pre-processing and basic editing - Histogram Equalization, Window and Level, Erosion*, Dilation*, Brightness, Warmness, Coolness, Blur, Sharpen, Rotate
- Saving images
- Printing images
• Keyboard Shortcuts

• Comparing images side by side

• Acquiring pixel intensities at user selected points

* denotes that the feature/operation has not yet been implemented.

### A.2 Installation and Running Instructions

SKIPT can be downloaded from the software section of NCSU’s Image Analysis Laboratory. Based on the user’s operating system, the corresponding version should be downloaded. SKIPT has been tested and verified to work on the following platforms:

• Mac OS X (Version 10.8 Mountain Lion) and higher

• Fedora 18 and higher

• Ubuntu 12.04 LTS and higher

• Red Hat Enterprise Linux 6 and higher

It should work on any recent Linux distribution.

For running SKIPT, the user simply clicks on the executable (downloaded from the source mentioned above) and starts working. No additional libraries or external dependencies are required.

### A.3 Working with SKIPT

The algorithms supported by SKIPT are divided into 6 categories according to type, nature, and use - **Image Pre-Processing**, **Edge Detectors**, **Image Derivatives**, **Feature Detectors**, **Image Segmentation**, and **Miscellaneous Operations**. Such segregation makes it easy to find the algorithm needed.

In addition to these 6 categories, the menu bar also contains a File Menu and a Zoom Menu.
A.3.1 File Menu

The File Menu contains options to:

1. Opening images
2. Switching to Block Diagram Mode
3. Printing images
4. Saving images
5. Exit application

A.3.1.1 Opening Images

Every operation begins with the loading of an image into the SKIPT environment. The user clicks the File menu and selects Open. Two windows appear - parent window and child window. Parent window displays the original image, while the child window displays the updated image after every operation. The first time the user an image file, both windows have the same image.
A.3.1.2 Switching to Block Diagram Mode

The “show stopper” of SKIPT is the ability to switch to a block diagram based image processing, algorithm simulation, and evaluation environment with a press of a button. The user simply clicks on Switch to Block Diagram Mode from the File menu and is transferred to an interactive model-based design environment. Figure A.3 shows the block diagram environment. In this environment all the operations are represented by ‘functional blocks’. A model is created by logically linking blocks together. It can be thought of as an executable file that the user can continuously refine throughout the development process. Each block has associated parameters which the user can adjust accordingly (future work), satisfying the exploratory as well as the performance aspect of the concerned operation. When the user completes a block design, the system also generates an actual code. Thus, merely through block diagrams and without the need of writing extensive codes, the system provides an active analysis and implementation of various image processing/computer vision algorithms.

Every model starts by placing the Open (to load the image which gets displayed on a side window), followed by any number of blocks. The blocks can be moved anywhere on the canvas. Once the blocks have been placed, they are “double clicked”. This is a way of letting the system know that the concerned operation is placed on an operations queue. The Execute is placed last, marking the end of every model. Finally, the Execute button is double clicked, and depending upon the blocks used to form the operations queue, a program...
is generated and executed. The output of the whole operation is displayed on a second side window. The first window (containing the original image) is also visible alongside the second window for analysis purposes.

A.3.2 **Image Pre-Processing Menu**

Image pre-processing involves operations to enhance image data and/or or suppress undesired distortions prior to any computational processing. These operations do not increase the image information, yet they come with a risk of emphasizing image artifacts and loss of information if not used correctly.
Name of Operation: Histogram Equalization

Purpose: Adjusting and stretching image intensity range to enhance contrast. It uses a mono-
non-linear mapping to change the intensity values of the input image's pixels in such a way that the output image's pixels have a uniformity in the distribution of intensities.

Parameters changeable by user: None

Figure A.4 Histogram Equalization operation in action. From the Image Pre-Processing Menu, the user clicks on Histogram Equalization and obtains the equalized image shown in the child window.
Name of Operation: Window and Level

Purpose: Contrast expansion of image pixels within a given window range. There are two parameters which control the pixel intensities - the Window which is the width of the range, and the Level which is the mid-gray brightness value of the window.

Parameters changeable by user:
1. Lower threshold limit
2. Upper threshold limit

Figure A.5 Window and Level Operation. Pixel values darker than the lower threshold value are mapped to black (value 0), while pixels brighter than the upper threshold value are mapped to white (value 255). All pixel values in between the two thresholds are scaled linearly in 0-255 range.

Name of Operation: Brightness

Purpose: To vary the degree of lightness (color closer to white) or darkness (color closer to black) in the image.

Parameters changeable by user: Slider to increment/decrement the value of 3 channels (RGB).
Figure A.6 Brightness operation in action. The slider is used to vary the amount of lightness/darkness in the image. Warmness and Coolness operations are also similar.

Name of Operation: Warmness
Purpose: To vary the image colors closeness to yellow.
Parameters changeable by user: Slider to increment/decrement the value of Red and Green channels.

Name of Operation: Coolness
Purpose: To vary the image colors closeness to blue.
Parameters changeable by user: Slider to increment/decrement the value of Blue channel.

Name of Operation: Smoothing/Blur
Purpose: Smoothing [Fer][Webf] consists of convolving the image with a blurring kernel. The intent is to smooth the image and make it appear ‘blurry’. As a result, the hard edges in the image are softened, and the overall spatial frequency is lowered.
Parameters changeable by user: Value of Gaussian sigma used for smoothing.
Figure A.7 Smoothing operation in action. The slider is used to change the value of Gaussian Sigma, varying the amount of blurring in the image.

Name of Operation: Sharpen
Purpose: “Sharpness can be defined as edge contrast, that is the contrast along the edges in a picture” [Weba]. The idea is to highlight details in an image without introducing noise or artifacts.
Parameters changeable by user: None

Name of Operation: Grayscale
Purpose: A grayscale image is an image in which the value of each pixel is a single sample, that is, it carries only intensity information.
Parameters changeable by user: None

Name of Operation: Inverted Grayscale
Purpose: The complement of a grayscale image is an inverted grayscale image.
Parameters changeable by user: None
Figure A.8 Grayscale operation in action. From the Image Pre-Processing Menu, the user clicks on Grayscale and obtains the monotone image shown in the child window. Inverted Grayscale and Log are also similar.

<table>
<thead>
<tr>
<th>Name of Operation:</th>
<th>Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose:</td>
<td>Log colormap converts pixel values to a logarithmic scale. In such a scheme, the values represented by colors increases exponentially.</td>
</tr>
<tr>
<td>Parameters changeable by user:</td>
<td>None</td>
</tr>
</tbody>
</table>

A.3.3 Zoom Menu

The Zoom Menu contains options to change the visual size of the image. It includes operations like Zoom In (upscaling the image), Zoom Out (downscaling the image), Normal Size, and Fit-to-Window.

A.3.4 Derivatives Menu

Image derivatives helps in identifying regions of discontinuities in an image. They correspond to a directional change in the intensity or color of image.
The following functions find the image derivatives by calculating the difference between the front pixel and rear pixel in the respective direction.

**Name of Operation:** First Derivative X Direction  
**Purpose:** Vertical Edges  
**Parameters changeable by user:** None

![Figure A.9 First Derivative X Direction Operation](image)

**Name of Operation:** First Derivative Y Direction  
**Purpose:** Horizontal Edges  
**Parameters changeable by user:** None

**Name of Operation:** Second Derivative X Direction  
**Purpose:** Vertical Edges  
**Parameters changeable by user:** None
Name of Operation: Second Derivative Y Direction
Purpose: Horizontal Edges
Parameters changeable by user: None

The following functions find the image derivatives by calculating the respective derivative of an isotropic, zero mean Gaussian function, and applying it to the image.

Name of Operation: First Derivative X Direction Gaussian
Purpose: Vertical Edges
Parameters changeable by user: Gaussian Sigma

Name of Operation: First Derivative Y Direction Gaussian
Purpose: Horizontal Edges
Parameters changeable by user: Gaussian Sigma

Figure A.10 First Derivative Y Direction Gaussian Operation. The slider allows the user to vary the value of sigma used for calculating the derivative.
Name of Operation: Second Derivative X Direction Gaussian
Purpose: Vertical Edges
Parameters changeable by user: Gaussian Sigma

Name of Operation: Second Derivative Y Direction Gaussian
Purpose: Horizontal Edges
Parameters changeable by user: Gaussian Sigma

A.3.5 Edge Detectors Menu

Edge detection [Lin98] [ZT98] is the process of identifying points of sharp brightness changes and discontinuities in an image. The discontinuities can be sudden changes in pixel intensities, depth, surface orientation, thereby describing the object boundaries in a scene.

There are many algorithms for finding edges in an image, but the ones used in SKIPT are grouped into two categories - ‘edge detection methods’ and ‘edge strength methods’. SKIPT has edge strength operators like Sobel Operator, Prewitt Operator, Robert Cross Operator, Laplacian, and Laplacian of Gaussian. Under the ‘edge detection methods’, Canny has been implemented.

Edge strength operators calculates the gradient of image intensity at each point. Gradient of an image is a 2-dimensional vector. The gradient along the horizontal direction gives the vertical edges, while the gradient along the vertical direction gives the horizontal edges. These can then be combined together to find the absolute magnitude of the gradient at each point and the orientation of that gradient. Thresholding the gradient gives the edges of the object. A greater difference in intensity corresponds to a higher magnitude of derivative and thus gives a clearer edge.

Canny edge detector [Can86] [Gre] is a ‘true edge detector’. It gives a yes or no decision for the presence of the edge. It has a low error rate, the edge points obtained are well localized, and it gives only one response to a single edge.

The operations under the Edge Detectors Menu and their corresponding parameters changeable by user are discussed below.
Name of Operation: Sobel Operator
Purpose: First order edge strength operator which combines Gaussian smoothing and differentiation. The operator consists of a pair of $3 \times 3$ convolution kernels.
Parameters changeable by user: Edge strength threshold value

Figure A.11 Edge Detection using Sobel Operator. The slider is used to vary the Threshold value.

Name of Operation: Prewitt Operator
Purpose: First order edge strength operator used for edge detection
Parameters changeable by user: Edge strength threshold value

Name of Operation: Robert-Cross Operator
Purpose: First order edge strength operator used for edge detection
Parameters changeable by user: Edge strength threshold value
Name of Operation: Laplacian
Purpose: “The Laplacian is a 2-D isotropic measure of the second spatial derivative of an image” [RFW]. It works by emphasizing those regions which vary greatly in intensity from their immediate neighborhood.
Parameters changeable by user: Edge strength threshold value

Name of Operation: Laplacian of Gaussian
Purpose: If a Laplacian kernel is applied to an image on which a Gaussian smoothing kernel has already been applied, then it is called Laplacian of Gaussian (LoG) [Gun99] [Wan]. The idea is to reduce the sensitivity to noise.
Parameters changeable by user: Edge strength threshold value

Name of Operation: Canny
Purpose: Canny edge detector is a ‘true edge detector’ as it gives a yes or no decision for the presence of the edge. It has a low error rate, the edge points obtained are well localized, and it gives only one response to a single edge.
Parameters changeable by user: 1. Upper threshold value
2. Lower threshold value
Figure A.12 Canny Edge Detection Operation. It uses two thresholds. Because of noise, using a single threshold has its limitations. A high threshold removes significant information, while a low threshold leads to generating numerable false edge points. To avoid this, 2 thresholds - \( T_{\text{upper}} \) and \( T_{\text{lower}} \) are used. Not only does this reduces false edge points in the output image, but also links the edges and reduces discontinuities.

A.3.6 Feature Detectors Menu

A feature is a “point of interest”. Technically, “features correspond to a sparse set of local measurements that capture the essence of the underlying input images and that encode their interesting structure” [GL11].

Harris Detector and Harris Laplace Detector have been implemented in SKIPT.

**Name of Operation:** Harris Detector

**Purpose:** Harris Detector [HS88] uses a gradient formulation to detect changes in intensity over a window/region, and return keypoints. These keypoints correspond to corner points of an image.

**Parameters changeable by user:**
1. Sigma value
2. Harris point threshold value
Name of Operation: Harris Laplace Detector

Purpose: “The Harris-Laplace detector uses a scale-adapted Harris function to localize points in space and then selects the points for which the Laplacian of Gaussian attains a maximum over scale” [MS02] [MS04]. This formulation makes the detected points invariant to scale.

Parameters changeable by user:
1. Sigma value
2. Harris point threshold value
3. Laplace point threshold value.

Figure A.13 Harris Laplace Operation in action. There are three user-defined inputs - sigma, threshold value for Harris points, and threshold value for Laplacian scale space. The feature points along with their corresponding scale are listed in a text file.

A.3.7 Image Segmentation Menu

Image segmentation is the technique of partitioning an image into distinct regions, and grouping similar pixels together. The purpose is to provide a meaningful, useful, and easier representation of image analysis.
**Color Based Segmentation** and **Watershed Segmentation** have been implemented in SKIPT.

<table>
<thead>
<tr>
<th>Name of Operation:</th>
<th>Color Based Segmentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose:</td>
<td>Segmentation based on <em>partitioning</em> of the color space [Webd]. The user selects a dominant/reference color ((R_0, G_0, B_0)) which he/she wants to extract from the image ((f(x,y))), and from every pixel color in image ((R(x,y), G(x,y), B(x,y))), a Cartesian distance is taken from the reference color.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters changeable by user:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Red threshold value</td>
</tr>
<tr>
<td>2. Green threshold value</td>
</tr>
<tr>
<td>3. Yellow threshold value</td>
</tr>
<tr>
<td>4. Binary sphere threshold (radius) value</td>
</tr>
</tbody>
</table>

\[
g(x, y) = \begin{cases} 1 &  d(x, y) \leq T_{\text{val}} \\ 0 &  d(x, y) > T_{\text{val}} \end{cases} \]

\[
d(x, y) = \sqrt{(R(x, y) - R_0)^2 + (G(x, y) - G_0)^2 + (B(x, y) - B_0)^2}
\]

Here \(g(x, y)\) is the value of the pixels of the resultant binary image. Based on the thresholding value, regions in the input test image bearing similar tones as that of the reference color are highlighted in black (or white) in the output binary image.

<table>
<thead>
<tr>
<th>Name of Operation:</th>
<th>Watershed Segmentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose:</td>
<td>The watershed of an ‘edgeness image’ can be used for segmentation. It correspond to objects, and shows object boundaries.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters changeable by user:</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
</tr>
</tbody>
</table>
**Figure A.14** Color Based Segmentation in SKIPT. A reference color is taken from which Cartesian distances are taken at every pixel color. This distance is then thresholded. For instance, if the user wants to segment yellow colors in the image then the red channel and green channel are made high. The thresholding value is correspondingly adjusted so as to bring the region of interest inside the 'sphere in RGB space', centered on the reference color.

### A.3.8 Miscellaneous Operations Menu

<table>
<thead>
<tr>
<th>Name of Operation:</th>
<th>Line Profile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose:</td>
<td>A line profile [Cor] [Nur] is a graph showing the variation of pixel intensities along a line for an image. It returns the grayscale values of the pixels along a line.</td>
</tr>
</tbody>
</table>

**Parameters changeable by user:** Points along which the profile has to be obtained.

<table>
<thead>
<tr>
<th>Name of Operation:</th>
<th>Image Histogram</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose:</td>
<td>A graph of the total number of pixels at each intensity/color level, histogram shows the tonal range (brightness value) of an image. Analyzing the histogram data can provide useful insight in attaining better quality images by adjusting image conditions [Cor].</td>
</tr>
</tbody>
</table>

**Parameters changeable by user:** None.