5.1 Introduction

This chapter includes implementation of edge detection method, Quadtree decomposition method, watershed segmentation method results with statistical information and graphs.

A typical edge might be (for instance) the border between a block of red color and a block of yellow; in contrast a line can be a small number of pixels of a different color on an otherwise unchanging background. There will be one edge on each side of the line. Edges play quite an important role in all applications of image processing.

5.2 Edge Detection using first and second order Derivatives

Most of the edge detection operators for image segmentation are lies in the 1\textsuperscript{st} order edge detection method, where most of the operators are operated on minimum threshold for edge detection and that operators are failed to detect the edge on the higher threshold. Hence it is necessary to define an operator that will work on higher threshold to detect the edge and the proposed test operator as discussed in chapter 4, works best on higher thresholds value as shown in
following table, and the results are compared with all other 1\textsuperscript{st} order and 2\textsuperscript{nd} order edge detection operators.

These are following major points are considered while implementing the concepts:

1. Find out the maximum edges using various image segmentation techniques.
2. Compared the results of edge detection techniques so as to determine the flexibility of edge detection methods.

Input Image for comparison of the edges with 1\textsuperscript{st} order, 2\textsuperscript{nd} order edge detection with test operator using matlab tool.

5.2.1 Edge Detection Process

1. Input: Image ‘I’ of size [M,N]
2. Output: Edge from Image ‘I’ of size [M,N]

In this research work various garyscale images has been taken in to consideration for the experiments, out of the set two sample images with the results are given below.

Figure (5.1) Cameraman.tif [256,256] Input to Edge detection process.
Output is generated using Edge Detection Process by using Operators. Following table shows the comparative result of all standard operator with Test operator as discussed in chapter 3 and chapter 4.

Sample Set 1
Input: cameraman.tif
Output: camreaman.tif containing Edges. With variable threshold

<table>
<thead>
<tr>
<th>Threshold</th>
<th>Canny Operator</th>
<th>Test Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.01</td>
<td><img src="image1" alt="Canny Operator 0.01" /></td>
<td><img src="image2" alt="Test Operator 0.01" /></td>
</tr>
<tr>
<td>0.05</td>
<td><img src="image3" alt="Canny Operator 0.05" /></td>
<td><img src="image4" alt="Test Operator 0.05" /></td>
</tr>
<tr>
<td>0.09</td>
<td><img src="image1.png" alt="Image" /></td>
<td><img src="image2.png" alt="Image" /></td>
</tr>
<tr>
<td>------</td>
<td>---------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>0.3</td>
<td><img src="image3.png" alt="Image" /></td>
<td><img src="image4.png" alt="Image" /></td>
</tr>
<tr>
<td>0.7</td>
<td><img src="image5.png" alt="Image" /></td>
<td><img src="image6.png" alt="Image" /></td>
</tr>
<tr>
<td>Threshold</td>
<td>Sobel</td>
<td>Test Operator</td>
</tr>
<tr>
<td>-----------</td>
<td>-------</td>
<td>---------------</td>
</tr>
<tr>
<td>0.8</td>
<td><img src="image1.png" alt="Image" /></td>
<td><img src="image2.png" alt="Image" /></td>
</tr>
<tr>
<td>0.9</td>
<td><img src="image3.png" alt="Image" /></td>
<td><img src="image4.png" alt="Image" /></td>
</tr>
<tr>
<td>0.01</td>
<td><img src="image5.png" alt="Image" /></td>
<td><img src="image6.png" alt="Image" /></td>
</tr>
<tr>
<td>Threshold</td>
<td>Description</td>
<td>Image</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
<td>-------</td>
</tr>
<tr>
<td>0.7</td>
<td>The Result is same as the result obtained in 0.7 Threshold</td>
<td><img src="image1.png" alt="Image" /></td>
</tr>
<tr>
<td>0.8</td>
<td>The Result is same as the result obtained in 0.7 Threshold</td>
<td><img src="image2.png" alt="Image" /></td>
</tr>
<tr>
<td>0.9</td>
<td>The Result is same as the result obtained in 0.7 Threshold</td>
<td><img src="image3.png" alt="Image" /></td>
</tr>
</tbody>
</table>

Sample Set 2  
Input: Tire.tif  
Output: Tire.tif containing Edges. With variable threshold
Figure (5.2) Tire.tif [256,256] Input to Edge Detection Process

<table>
<thead>
<tr>
<th>Threshold</th>
<th>Canny Operator</th>
<th>Test Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.05</td>
<td><img src="image1.jpg" alt="Image" /></td>
<td><img src="image2.jpg" alt="Image" /></td>
</tr>
<tr>
<td>0.1</td>
<td><img src="image3.jpg" alt="Image" /></td>
<td><img src="image4.jpg" alt="Image" /></td>
</tr>
<tr>
<td>0.5</td>
<td><img src="image5.jpg" alt="Image" /></td>
<td><img src="image6.jpg" alt="Image" /></td>
</tr>
</tbody>
</table>
### 5.2.2 Discussion and Interpretation

Cameraman.tif and Tire.tif has taken as a sample images for the experiment. Proposed ‘Test’ operator is applied on the said images at various thresholds and compared with the previous known operators like: Gaussian, Prewit, Sobel, Canny, roberts, etc. we found the operation of the “Test” operator is best among the mentioned operators after visually interpretation.

For example, on the image ‘cameraman.tif’, canny operator is applied from threshold 0.05 to 0.9 and comparitively on the same image a “Test” operator is applied.
a) At 0.05 threshold “Test” operator gave more edges than that of the canny and sobel operator.

b) Generally, When we increase the threshold, the weak edges gets disapper , but using “ Test “ operator we got maximum edges upto the highest threshold i.e. 0.9. As compare to previously known operators.

c) The results are shown in section (5.2.1)

5.3 Region Based Methods

Region Based Segmentation methods has objective to partition an image into regions. The basic formulations are for regions as per Equation (1) [1][3]

Here, P(Ri) is the logical predicate defined over the points in the set Ri and Ø is the null set, the condition (a) indicates that the segmentation must be complete that is every pixel must be in region, (b) indicates that point in the region must be connected in some predefined sense, (c) indicates that the regions must be disjoints, (d) indicates that, this deals with the properties that must be satisfied by the pixel in a segmented region that is P(Ri)=TRUE if all pixel in Rj have the same gray level and (e) indicates that the region Ri and Rj are different in the sense of predicate P.[5]

5.3.1 Region growing and merging

As its name indicates, region growing is procedure that groups pixel or sub regions into larger regions based on predefined criteria. The basic approach is to start with the set of “seed” points and from these grow regions by appending to
each seed those neighboring pixels that have properties similar to the seed. [6] The selection of similarity criteria depends not only on the problem under considerations but also on the type of image that available. The use of connectivity was the fundamental in solving the above problem. [5]

5.3.2 Region splitting and merging using quadtree

This is an alternative method to subdivide an image initially into set of arbitrary, disjointed regions and then merge and/or split the regions in an attempt to satisfy the conditions for region formation as per eq (1). By splitting and merging algorithm that iteratively works towards satisfying these constrains of similarity [7].

Let X represents the entire image region and selects the predicate P. One approach for segmenting X is to subdivide it successively into smaller and smaller quadrant regions so that for any region Xi, P (Xi) = TRUE. Hence start with the entire region. If P(X) = FALSE, divide the image into quadrants. If P is FALSE for any quadrant, subdivide that quadrant into sub quadrants, and so on. This particular splitting techniques has a convenient representation in the form of a so called quad tree [1, 6]

Quadtree decomposition is used for edge detection with respect to the region growing and merging. While applying a quadtree the input image must be in gray scale and the size of image must be in power of 2, this will certainly helps in decomposition of an image into 2n regions. The regions were merged as per Equation (6). The edge detection is an important issue for complete understanding
of image; hence building the regions is the simplest from of image analysis and image segmentation. Quadtree is an image segmentation method basically used for region splitting and merging; based on the criteria of similarity were it prepares the segment by applying the predicate (Threshold) on image. As the predicate is increased progressively, the blocks containing the edges by Quadtree are less with respect to the blocks contains the edges at low threshold. This is due to the merging of regions by Quadtree.

As the images acquired from the problem domain are need not to be in grayscale or need not be the square so modification must be required in the quadtree so as it has to be applied on the input image and must prepare the regions for the same. Quadtree is modified by incorporating preprocessing on input image by converting source image into grayscale and size of source image is resized (if necessary) in the power of 2. Therefore the improved algorithm of quadtree is as follows

**Algorithm**

**Quadtree (Threshold T, NoRegions N)**

1. For all N Regions, Split each Ni into four disjoints quadrants Ni (Xj), for which T (Ni (Xj)) =FALSE

2. Merge any adjacent Regions Ni (Xj) and Ni (Xk) for T (Ni (Xj) U Ni (Xk)) = TRUE

3. Stop when no further merging or splitting of region N is possible.

With the help of this algorithm we can easily specify the total number of regions and threshold. As the threshold is increased progressively the edges generated
using quadtree are minimum due to the merging of regions. The results are obtained using quadtree algorithm are shown in following figures and related statistical information is placed in tables. The statistical comparative analysis is shown in the graphs.
Figure (5.7) Tree.tif with its histogram

Figure (5.8) Quadtree at $2^4$ Regions with 0.25 Threshold

Figure (5.9) rice.png with its histogram

Figure (5.10) Quadtree at $2^4$ Regions with 0.05 Threshold

Figure (5.11) pout.tif with its histogram

Figure (5.12) Quadtree at $2^4$ Regions with 0.20 Threshold
### Table 5.2: Threshold vs. Number of Blocks containing Edges

<table>
<thead>
<tr>
<th>Threshold</th>
<th>Total Number of Regions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>eight.tif</td>
</tr>
<tr>
<td>0.5</td>
<td>1509</td>
</tr>
<tr>
<td>0.45</td>
<td>2041</td>
</tr>
<tr>
<td>0.4</td>
<td>2552</td>
</tr>
<tr>
<td>0.35</td>
<td>3266</td>
</tr>
<tr>
<td>0.3</td>
<td>3971</td>
</tr>
<tr>
<td>0.25</td>
<td>4994</td>
</tr>
<tr>
<td>0.2</td>
<td>6371</td>
</tr>
<tr>
<td>0.15</td>
<td>8323</td>
</tr>
<tr>
<td>0.1</td>
<td>11063</td>
</tr>
<tr>
<td>0.05</td>
<td>16117</td>
</tr>
</tbody>
</table>

Graph 5.1: Graph represents Threshold vs. Total Number of Regions
Table 5.3: Threshold vs. Number of Blocks containing Edges

<table>
<thead>
<tr>
<th>Threshold</th>
<th>total_number_of_region_containing_edges</th>
<th>eight.tif</th>
<th>tire.tif</th>
<th>trees.tif</th>
<th>rice.png</th>
<th>pout.tif</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td></td>
<td>100</td>
<td>232</td>
<td>0</td>
<td>0</td>
<td>144</td>
</tr>
<tr>
<td>0.45</td>
<td></td>
<td>268</td>
<td>536</td>
<td>16</td>
<td>0</td>
<td>304</td>
</tr>
<tr>
<td>0.4</td>
<td></td>
<td>428</td>
<td>772</td>
<td>184</td>
<td>0</td>
<td>356</td>
</tr>
<tr>
<td>0.35</td>
<td></td>
<td>828</td>
<td>932</td>
<td>532</td>
<td>32</td>
<td>480</td>
</tr>
<tr>
<td>0.3</td>
<td></td>
<td>1232</td>
<td>1164</td>
<td>1012</td>
<td>800</td>
<td>484</td>
</tr>
<tr>
<td>0.25</td>
<td></td>
<td>1964</td>
<td>1788</td>
<td>1804</td>
<td>3388</td>
<td>544</td>
</tr>
<tr>
<td>0.2</td>
<td></td>
<td>3120</td>
<td>2980</td>
<td>3308</td>
<td>5688</td>
<td>680</td>
</tr>
<tr>
<td>0.15</td>
<td></td>
<td>4980</td>
<td>5492</td>
<td>5500</td>
<td>7612</td>
<td>920</td>
</tr>
<tr>
<td>0.1</td>
<td></td>
<td>7776</td>
<td>9248</td>
<td>10332</td>
<td>10412</td>
<td>1392</td>
</tr>
<tr>
<td>0.05</td>
<td></td>
<td>12784</td>
<td>16364</td>
<td>25784</td>
<td>27376</td>
<td>4228</td>
</tr>
</tbody>
</table>

Graph 5.2: Graph represents Threshold vs. Total Number of Regions containing the Edges
<table>
<thead>
<tr>
<th>Threshold</th>
<th>Total Number of Region Merged</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>eight.tif</td>
</tr>
<tr>
<td>0.5</td>
<td>1409</td>
</tr>
<tr>
<td>0.45</td>
<td>1773</td>
</tr>
<tr>
<td>0.4</td>
<td>2104</td>
</tr>
<tr>
<td>0.35</td>
<td>2438</td>
</tr>
<tr>
<td>0.3</td>
<td>2739</td>
</tr>
<tr>
<td>0.25</td>
<td>3030</td>
</tr>
<tr>
<td>0.2</td>
<td>3251</td>
</tr>
<tr>
<td>0.15</td>
<td>3343</td>
</tr>
<tr>
<td>0.1</td>
<td>3297</td>
</tr>
<tr>
<td>0.05</td>
<td>3333</td>
</tr>
</tbody>
</table>

Table 5.4: Threshold vs. Number of Blocks Merged

Graph 5.2: Graph represents Threshold vs. Total Number of Regions Merged.
5.4 Design Code

5.4.1 Edge Detection Procedure

function varargout = EdgeDetection(varargin)

% EDGEDETECTION M-file for EdgeDetection.fig
% EDGEDETECTION, by itself, creates a new EDGEDETECTION or raises
the existing
% singleton*.
% H = EDGEDETECTION returns the handle to a new EDGEDETECTION
or the handle to
% the existing singleton*.
% EDGEDETECTION('CALLBACK',hObject,eventData,handles,...) calls the
local
% function named CALLBACK in EDGEDETECTION.M with the given
input arguments.
% EDGEDETECTION('Property','Value',...) creates a new
EDGEDETECTION or raises the
% existing singleton*. Starting from the left, property value pairs are
% applied to the GUI before EdgeDetection_OpeningFunction gets called.
An
% unrecognized property name or invalid value makes property application
% stop. All inputs are passed to EdgeDetection_OpeningFcn via varargin.
% *See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one
instance to run (singleton)".
% See also: GUIDE, GUIDATA, GUIDATAHANDLES
% Copyright 2002-2003 The MathWorks, Inc.
% Edit the above text to modify the response to help EdgeDetection
% Last Modified by GUIDE v2.5 22-Oct-2005 15:14:26
% Begin initialization code - DO NOT EDIT

gui_Singleton = 1;
gui_State = struct('gui_Name', mfilename, ...
    'gui_Singleton', gui_Singleton, ...
    'gui_OpeningFcn', @EdgeDetection_OpeningFcn, ...
    'gui_OutputFcn', @EdgeDetection_OutputFcn, ...
    'gui_LayoutFcn', [], ...
    'gui_Callback', []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
% --- Executes just before EdgeDetection is made visible.
function EdgeDetection_OpeningFcn(hObject, eventdata, handles, varargin)

% This function has no output args, see OutputFcn.

% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to EdgeDetection (see VARARGIN)
% Choose default command line output for EdgeDetection

handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes EdgeDetection wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.

function varargout = EdgeDetection_OutputFcn(hObject, eventdata, handles)

% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Get default command line output from handles structure

varargout{1} = handles.output;

function img_Callback(hObject, eventdata, handles)
% hObject    handle to img (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of img as text
%        str2double(get(hObject,'String')) returns contents of img as a
double
% --- Executes during object creation, after setting all properties.

function img_CreateFcn(hObject, eventdata, handles)
% hObject    handle to img (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.

if ispc
    set(hObject,'BackgroundColor','white');
else

set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'))
;
end

% --- Executes on button press in pushbutton1.

function pushbutton1_Callback(hObject, eventdata, handles)

% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

x=char(get(handles.img,'String'))
i=imread(x);
axes(handles.axes1);
imshow(i);

% --- Executes on button press in pushbutton2.

function pushbutton2_Callback(hObject, eventdata, handles)

% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Author : Ashok T. Gaikwad

% Purpose : Use of Test Operator for Edge Detection and comprision of Test Operator with the standard Operators

\[x = \text{char}(\text{get(handles.img,'String'))}\]
\[i = \text{imread}(x);\]
\[\text{val} = \text{get(handles.popup1,'Value');}\]
\[\text{thr} = \text{get(handles.popup2,'Value');}\]
\[\text{sz} = \text{size}(i);\]
\[\text{asz} = \text{size}(\text{sz});\]
\[\text{axes(handles.axes3);}\]

% To Convert color image to gray
\[\text{if asz(1,2)==3}\]
\[j = \text{rgb2gray}(i);\]
\[i = j;\]
\[\text{end}\]

% Select the Options for threshold
\[\text{if thr>0}\]
\[\text{switch thr}\]
\[\text{case 1}\]
\[\text{thrval}=0.01\]
case 2
  thrval=0.02

case 3
  thrval=0.03

case 4
  thrval=0.04

case 5
  thrval=0.05

case 6
  thrval=0.06

case 7
  thrval=0.07

case 8
  thrval=0.08

case 9
  thrval=0.09

case 10
  thrval=0.1

case 11
  thrval=0.2

case 12
  thrval=0.3

case 13
thrval=0.4

case 14
    thrval=0.5

case 15
    thrval=0.6

case 16
    thrval=0.7

case 17
    thrval=0.8

case 18
    thrval=0.9

case 19
    thrval=1

end

switch val
    case 1
        ed=edge(i,'canny',thrval);
    case 2
        ed=edge(i,'prewitt',thrval);
    case 3
        ed=edge(i,'sobel',thrval);
    case 4
ed=edge(i,'roberts',thrval);

case 5
ed=edge(i,'log',0);

case 6
ed=edge(i,'zerocross',0);

case 7
i=medfilt2(i); %noise reduction using the median filter
ed=edge(i,'ashok',thrval,'both');

end

else

switch val

case 1
ed=edge(i,'canny');

case 2
ed=edge(i,'prewitt');

case 3
ed=edge(i,'sobel');

case 4
ed=edge(i,'roberts');

case 5
ed=edge(i,'log',0);

case 6
ed=edge(i,'zerocross',0);
end

if val==7
    ed=medfilt2(ed);
    colormap('gray');
    imshow(ed);
else
    imshow(ed);
end

% --- Executes on selection change in popupmenu1.

function popupmenu1_Callback(hObject, eventdata, handles)

    % hObject    handle to popupmenu1 (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    structure with handles and user data (see GUIDATA)

    %figure,imhisteq(ed);
% Hints: contents = get(hObject,'String') returns popupmenu1 contents as
cell array

% contents{get(hObject,'Value')} returns selected item from
popupmenu1
val = get(hObject,'Value');

% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)
    % hObject    handle to popupmenu1 (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    empty - handles not created until after all CreateFcns called

    % Hint: popupmenu controls usually have a white background on
    Windows.
    % See ISPC and COMPUTER.
    if ispc
        set(hObject,'BackgroundColor','white');
    else

        set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'))
    ;
    end
function edit2_Callback(hObject, eventdata, handles)

% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit2 as text
%        str2double(get(hObject,'String')) returns contents of edit2 as a double

% --- Executes during object creation, after setting all properties.

function edit2_CreateFcn(hObject, eventdata, handles)

% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.

if ispc
    set(hObject,'BackgroundColor','white');
else
set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'))

end

% --- Executes on selection change in popup2.

function popup2_Callback(hObject, eventdata, handles)

% hObject    handle to popup2 (see GCBO)

% eventdata  reserved - to be defined in a future version of MATLAB

% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popup2 contents as cell
array
% contents{get(hObject,'Value')} returns selected item from popup2

% --- Executes during object creation, after setting all properties.

function popup2_CreateFcn(hObject, eventdata, handles)

% hObject    handle to popup2 (see GCBO)

% eventdata  reserved - to be defined in a future version of MATLAB

% handles    empty - handles not created until after all CreateFcns called
% Hint: popupmenu controls usually have a white background on Windows.

% See ISPC and COMPUTER.

if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'))
;
end

**EdgeDetectionfunction**

*function [eout,thresh] = edge(varargin)*

[a,method,thresh,sigma,H,kx,ky] = parse_inputs(varargin{:});

% Transform to a double precision intensity image if necessary
if ~isa(a, 'double')
    a = im2double(a);
end

m = size(a,1);
n = size(a,2);
rr = 2:m-1; cc=2:n-1;

% The output edge map:
e = repmat(false, m, n);

if strcmp(method,'canny')
    % Magic numbers
    GaussianDieOff = .0001;
    PercentOfPixelsNotEdges = .7; % Used for selecting thresholds
    ThresholdRatio = .4; % Low thresh is this fraction of the high.

    % Design the filters - a gaussian and its derivative

    pw = 1:30; % possible widths
    ssq = sigma*sigma;
    width = max(find(exp(-(pw.*pw)/(2*sigma*sigma))>GaussianDieOff));
    if isempty(width)
        width = 1; % the user entered a really small sigma
    end

    t = (-width:width);
    gau = exp(-(t.*t)/(2*ssq))/(2*pi*ssq); % the gaussian 1D filter
% Find the directional derivative of 2D Gaussian (along X-axis)
% Since the result is symmetric along X, we can get the derivative along
% Y-axis simply by transposing the result for X direction.

[x,y]=meshgrid(-width:width,-width:width);
dgau2D=-x.*exp(-(x.*x+y.*y)/(2*ssq))/(pi*ssq);

% Convolve the filters with the image in each direction
% The canny edge detector first requires convolution with
% 2D gaussian, and then with the derivative of a gaussian.
% Since gaussian filter is separable, for smoothing, we can use
% two 1D convolutions in order to achieve the effect of convolving
% with 2D Gaussian. We convolve along rows and then columns.

% smooth the image out
aSmooth=imfilter(a,gau,'conv','replicate');   % run the filter across rows
aSmooth=imfilter(aSmooth,gau,'conv','replicate'); % and then across columns

% apply directional derivatives
ax = imfilter(aSmooth, dgau2D, 'conv','replicate');
ay = imfilter(aSmooth, dgau2D', 'conv','replicate');

mag = sqrt((ax.*ax) + (ay.*ay));
magmax = max(mag(:));
if magmax>0
    mag = mag / magmax;  % normalize
end

% Select the thresholds
if isempty(thresh)
    [counts,x]=imhist(mag, 64);
    highThresh = min(find(cumsum(counts) >
PercentOfPixelsNotEdges*m*n)) / 64;
    lowThresh = ThresholdRatio*highThresh;
    thresh = [lowThresh highThresh];
elseif length(thresh)==1
    highThresh = thresh;
    if thresh>=1
        eid = sprintf('Images:%s:thresholdMustBeLessThanOne', mfilename);
        msg = 'The threshold must be less than 1.';
        error(eid, '%s', msg);
    end
    lowThresh = ThresholdRatio*thresh;
    thresh = [lowThresh highThresh];
elseif length(thresh)==2
    lowThresh = thresh(1);
highThresh = thresh(2);

if (lowThresh >= highThresh) | (highThresh >= 1)
    eid = sprintf('Images:%s:thresholdOutOfRange', mfilename);
    msg = 'Thresh must be [low high], where low < high < 1.,'
    error(eid,'%s',msg);
end

% The next step is to do the non-maximum supression.
% We will accrue indices which specify ON pixels in strong edgemap
% The array e will become the weak edge map.
idxStrong = [];
for dir = 1:4
    idxLocalMax = cannyFindLocalMaxima(dir,ax,ay,mag);
    idxWeak = idxLocalMax(mag(idxLocalMax) > lowThresh);
    e(idxWeak)=1;
    idxStrong = [idxStrong; idxWeak(mag(idxWeak) > highThresh)];
end

rstrong = rem(idxStrong-1, m)+1;
cstrong = floor((idxStrong-1)/m)+1;
e = bwselect(e, cstrong, rstrong, 8);
e = bwmorph(e, 'thin', 1);  % Thin double (or triple) pixel wide contours
elseif any(strcmp(method, {'log','marr-hildreth','zerocross'}))

% We don't use image blocks here

if isempty(H),
  fsize = ceil(sigma*3) * 2 + 1;  % choose an odd fsize > 6*sigma;
  op = fspecial('log',fsize,sigma);
else
  op = H;
end

op = op - sum(op(:))/prod(size(op)); % make the op to sum to zero
b = filter2(op,a);

if isempty(thresh)
  thresh = .75*mean2(abs(b(rr,cc)));
end

% Look for the zero crossings: +-, -+ and their transposes
% We arbitrarily choose the edge to be the negative point

[rx,cx] = find( b(rr,cc) < 0 & b(rr,cc+1) > 0 ... 
  & abs( b(rr,cc)-b(rr,cc+1) ) > thresh );  % [- +]
e((rx+1) + cx*m) = 1;

[rx,cx] = find( b(rr,cc-1) > 0 & b(rr,cc) < 0 ...
& abs( b(rr,cc-1)-b(rr,cc) ) > thresh );  \% [+ - ]
e((rx+1) + cx*m) = 1;

[rx,cx] = find( b(rr,cc) < 0 & b(rr+1,cc) > 0 ...
& abs( b(rr,cc)-b(rr+1,cc) ) > thresh);  \% [- +]'
e((rx+1) + cx*m) = 1;

[rx,cx] = find( b(rr-1,cc) > 0 & b(rr,cc) < 0 ...
& abs( b(rr-1,cc)-b(rr,cc) ) > thresh);  \% [+ -]'
e((rx+1) + cx*m) = 1;

% Most likely this covers all of the cases. Just check to see if there
% are any points where the LoG was precisely zero:
[rz,cz] = find( b(rr,cc)==0 );

if ~isempty(rz)
  % Look for the zero crossings: +0-, -0+ and their transposes
  % The edge lies on the Zero point
  zero = (rz+1) + cz*m;  \% Linear index for zero points
  zz = find(b(zero-1) < 0 & b(zero+1) > 0 ...
  & abs( b(zero-1)-b(zero+1) ) > 2*thresh);  \% [- 0 +]'
e(zero(zz)) = 1;

  zz = find(b(zero-1) > 0 & b(zero+1) < 0 ...
  & abs( b(zero-1)-b(zero+1) ) > 2*thresh);  \% [+ 0 -]'
e(zero(zz)) = 1;

  zz = find(b(zero-m) < 0 & b(zero+m) > 0 ...
& abs( b(zero-m)-b(zero+m) ) > 2*thresh);   % [- 0 +]
e(zero(zz)) = 1;

zz = find(b(zero-m) > 0 & b(zero+m) < 0 ... & abs( b(zero-m)-b(zero+m) ) > 2*thresh);   % [+ 0 -]
e(zero(zz)) = 1;
end

else  % one of the easy methods (roberts,sobel,prewitt)

% Determine edges in blocks for easy methods
nr = length(rr); nc = length(cc);

blk = bestblk([nr nc]);
nblks = floor([nr nc]/blk); nrem = [nr nc] - nblks.*blk;
mblocks = nblks(1); nblocks = nblks(2);
mb = blk(1); nb = blk(2);

if strcmp(method,'sobel')
    op = [-1 -2 -1;0 0 0;1 2 1]/8; % Sobel approximation to derivative
    x_mask = op';   % gradient in the X direction
    y_mask = op;

    scale = 4; % for calculating the automatic threshold
offset = [0 0 0 0];  % offsets used in the computation of the threshold

elseif strcmp(method,'prewitt')
    op = [-1 -1 -1;0 0 0;1 1 1]/6;  % Prewitt approximation to derivative
    x_mask = op';
    y_mask = op;

    scale = 4;
    offset = [0 0 0 0];

elseif strcmp(method, 'roberts')
    op = [1 0;0 -1]/sqrt(2);  % Roberts approximation to diagonal derivative
    x_mask = op;
    y_mask = rot90(op);

    scale = 6;
    offset = [-1 1 1 -1];

elseif strcmp(method,'ashok')
    op = [-2 0 2;-1 0 1;-2 0 2];
    %op = [-2 -1 -2;0 0 0;2 1 2];  % approximation to derivative
    x_mask = op';  % gradient in the X direction
    y_mask = op;
%scale = 4; % for calculating the automatic threshold

%offset = [-1 -1 -1 -1]; % offsets used in the computation of the threshold

scale = 8; % for calculating the automatic threshold

offset = [0 0 0 0]; % offsets used in the computation of the threshold

else

eid = sprintf('Images:%s:invalidEdgeDetectionMethod', mfilename);
msg = sprintf('%s %s', method, 'is not a valid method.');
error(eid, '%s', msg);
end

% compute the gradient in x and y direction
bx = abs(filter2(x_mask, a));
by = abs(filter2(y_mask, a));

% compute the magnitude
b = kx*bx.*bx + ky*by.*by;

% determine the threshold; see page 514 of "Digital Imaging Processing"
by

% William K. Pratt
if isempty(thresh), % Determine cutoff based on RMS estimate of noise
    % Mean of the magnitude squared image is a
    % value that's roughly proportional to SNR
    cutoff = scale*mean2(b(rr,cc));
    thresh = sqrt(cutoff);
else % Use relative tolerance specified by the user
    cutoff = (thresh).^2;
end

% compute the output image
rows = 1:blk(1);
for i=0:mblocks,
    if i==mblocks,
        rows = (1:nrem(1));
    end
    for j=0:nbblocks,
        if j==0,
            cols = 1:blk(2);
        elseif j==nbblocks,
            cols=(1:nrem(2));
        end
        if ~isempty(rows) & ~isempty(cols)
            r = rr(i*mb+rows); c = cc(j*nb+cols);
        end
    end
end
e(r,c) = (b(r,c)>cutoff) & ...
((bx(r,c) >= (kx*by(r,c)-eps*100)) & ...
(b(r+1,c-1) <= b(r,c)) & ...
(b(r,c) > b(r+1,c+1))) | ...
((by(r,c) >= (ky*bx(r,c)-eps*100)) & ...
(b(r-1,c+offset(1)) <= b(r,c)) & ...
(b(r,c) > b(r+1,c+offset(4)))));
end
end
end
end

if nargout==0,

imshow(e);
else
eout = e;
end

% Local Function : cannyFindLocalMaxima

% function idxLocalMax = cannyFindLocalMaxima(direction,ix,iy,mag);

[m,n,o] = size(mag);
% Find the indices of all points whose gradient (specified by the
% vector (ix,iy)) is going in the direction we're looking at.

switch direction
    case 1
        idx = find((iy<=0 & ix>-iy) | (iy>=0 & ix<-iy));
    case 2
        idx = find((ix>0 & -iy>=ix) | (ix<0 & -iy<=ix));
    case 3
        idx = find((ix<=0 & ix>iy) | (ix>=0 & ix<iy));
    case 4
        idx = find((iy<0 & ix<=iy) | (iy>0 & ix>=iy));
end

% Exclude the exterior pixels
if ~isempty(idx)
    v = mod(idx,m);
    extIdx = find(v==1 | v==0 | idx<=m | (idx>(n-1)*m));
    idx(extIdx) = [];
end

ixv = ix(idx);
iyv = iy(idx);
gradmag = mag(idx);

% Do the linear interpolations for the interior pixels
switch direction
    case 1
d = abs(iyv./ixv);
gradmag1 = mag(idx+m).*(1-d) + mag(idx+m-1).*d;
gradmag2 = mag(idx-m).*(1-d) + mag(idx-m+1).*d;
    case 2
d = abs(ixv./iyv);
gradmag1 = mag(idx-1).*(1-d) + mag(idx+m-1).*d;
gradmag2 = mag(idx+1).*(1-d) + mag(idx-m+1).*d;
    case 3
d = abs(ixv./iyv);
gradmag1 = mag(idx-1).*(1-d) + mag(idx-m-1).*d;
gradmag2 = mag(idx+1).*(1-d) + mag(idx+m+1).*d;
    case 4
d = abs(iyv./ixv);
gradmag1 = mag(idx-m).*(1-d) + mag(idx-m-1).*d;
gradmag2 = mag(idx+m).*(1-d) + mag(idx+m+1).*d;
end
idxLocalMax = idx(gradmag>=gradmag1 & gradmag>=gradmag2);
function \([I,\text{Method,\text{Thresh,\text{Sigma,H,\text{kx,ky}}]} = \text{parse\_inputs}(\text{varargin})\]

% OUTPUTS:

% I Image Data
% Method Edge detection method
% Thresh Threshold value
% Sigma standard deviation of Gaussian
% H Filter for Zero-crossing detection
% kx,ky From Directionality vector

error(nargchk(1,5,nargin,'struct'));

I = varargin{1};

checkinput(I,\{'\text{double}',\'\text{logical}',\'\text{uint8}',\'\text{uint16}'\},...\{'\text{nonsparse}',\'2d'\},mfilename,'I',1);

% Defaults
Method='sobel';
Thresh=[];
Direction='both';
Sigma=2;
H=[];
K=[1 1];

methods = {'canny','prewitt','sobel','marr-hildreth','log','roberts','zerocross','ashok'};
directions = {'both','horizontal','vertical'};

% Now parse the nargin-1 remaining input arguments

% First get the strings - we do this because the interpretation of the
% rest of the arguments will depend on the method.
nonstr = []; % ordered indices of non-string arguments
for i = 2:nargin
    if ischar(varargin{i})
        str = lower(varargin{i});
        j = strmatch(str,methods);
        k = strmatch(str,directions);
        if ~isempty(j)
            Method = methods{j(1)};
            if strcmp(Method,'marr-hildreth')
                wid = sprintf('Images:%s:obsoleteMarrHildrethSyntax', mfilename);
                msg = '''Marr-Hildreth'' is an obsolete syntax, use ''LoG'' instead.';
            end
        end
    end
end
warning(wid,'%s',msg);

end

elseif ~isempty(k)
    Direction = directions{k(1)};
else
    eid = sprintf('Images:%s:invalidInputString', mfilename);
    msg = sprintf('%s%s%s', 'Invalid input string: ''', varargin{i},'''.');
    error(eid,'%s',msg);
else
    nonstr = [nonstr i];
end
end

% Now get the rest of the arguments

eid_invalidArgs = sprintf('Images:%s:invalidInputArguments', mfilename);
msg_invalidArgs = 'Invalid input arguments';

switch Method
    case {'prewitt','sobel','roberts'}
threshSpecified = 0; % Threshold is not yet specified
for i = nonstr
    if prod(size(varargin{i}))<=1 & ~threshSpecified % Scalar or empty
        Thresh = varargin{i};
        threshSpecified = 1;
    elseif prod(size(varargin{i}))==2 % The dreaded K vector
        wid = sprintf('Images:%s:obsoleteKDirectionSyntax', mfilename);
        msg = sprintf('%s%s%s', 'BW = EDGE(... , K) is an obsolete syntax.',...
                      'Use BW = EDGE(... , DIRECTION),',...
                      ' where DIRECTION is a string.);
        warning(wid,'%s',msg);
        K=varargin{i};
    else
        error(eid_invalidArgs,msg_invalidArgs);
    end
end
% Use of Ashok with the Image Filter
case {'ashok'}
    %h=[0 -2 0; -2 8 -2; 0 -2 0];
    %i=imfilter(i,h,'conv','replicate');
    %i=imfilter(i,h);
    threshSpecified = 0; % Threshold is not yet specified
for i = nonstr
    if prod(size(varargin{i}))<=1 & ~threshSpecified % Scalar or empty
        Thresh = varargin{i};
        threshSpecified = 1;
    elseif prod(size(varargin{i}))==2  % The dreaded K vector
        wid = sprintf('Images:%s:obsoleteKDirectionSyntax', mfilename);
        msg = sprintf('%s%s%s', 'BW = EDGE(... , K) is an obsolete syntax. ',
                      'Use BW = EDGE(... , DIRECTION),',...
                      ' where DIRECTION is a string.');
        warning(wid,'%s',msg);
        K=varargin{i};
    else
        error(eid_invalidArgs,msg_invalidArgs);
    end
end

case 'canny'
    Sigma = 1.0; % Default Std dev of gaussian for canny
    threshSpecified = 0; % Threshold is not yet specified
    for i = nonstr
        if prod(size(varargin{i}))==2 & ~threshSpecified
            Thresh = varargin{i};
        end
    end
threshSpecified = 1;
elseif prod(size(varargin{i})) == 1
    if ~threshSpecified
        Thresh = varargin{i};
        threshSpecified = 1;
    else
        Sigma = varargin{i};
    end
elseif isempty(varargin{i}) & ~threshSpecified
    % Thresh = [];
    threshSpecified = 1;
else
    error(eid_invalidArgs,msg_invalidArgs);
end
end

case 'log'
    threshSpecified = 0; % Threshold is not yet specified
    for i = nonstr
        if prod(size(varargin{i})) <= 1 % Scalar or empty
            if ~threshSpecified
                Thresh = varargin{i};
                threshSpecified = 1;
            end
        end
    end
end
else
    \Sigma = \text{varargin}\{i\};
end
else
    \text{error(eid\_invalidArgs,msg\_invalidArgs);}  
end
end

\text{case 'zerocross'}
\text{threshSpecified = 0; \% Threshold is not yet specified}
\text{for } i = \text{nonstr}
\text{if prod(size(varargin\{i\}))}\leq 1 \& \sim\text{threshSpecified \% Scalar or empty}
    \text{Thresh = varargin}\{i\};
    \text{threshSpecified = 1;}
\text{elseif prod(size(varargin\{i\}))} > 1 \% The filter for zerocross
    \text{H = varargin}\{i\};
\text{else}
    \text{error(eid\_invalidArgs,msg\_invalidArgs);}  
\text{end}
\text{end}

\text{case 'marr-hildreth'}
\text{for } i = \text{nonstr}
if prod(size(varargin{i})) <= 1  % Scalar or empty
    Thresh = varargin{i};
elseif prod(size(varargin{i})) == 2  % The dreaded K vector
    wid = sprintf('Images:%s:dirFactorHasNoEffectOnMarrHildreth', mfilename);
    msg = 'The \[kx ky\] direction factor has no effect for ''Marr-Hildreth''.
    warning(wid,\%s\',msg);
elseif prod(size(varargin{i})) > 2 % The filter for zerocross
    H = varargin{i};
else
    error(eid_invalidArgs,msg_invalidArgs);
end
end

otherwise
    error(eid_invalidArgs,msg_invalidArgs);
end

if Sigma<=0
    eid = sprintf('Images:%s:sigmaMustBePositive', mfilename);
    msg = 'Sigma must be positive';
    error(eid,\%s\',msg);
end
switch Direction

case 'both',
    kx = K(1); ky = K(2);

case 'horizontal',
    kx = 0; ky = 1; % Directionality factor

case 'vertical',
    kx = 1; ky = 0; % Directionality factor

otherwise

    eid = sprintf('Images:%s:badDirectionString', mfilename);
    msg = 'Unrecognized direction string';
    error(eid,'%s',msg);

end
5.4.2 Quadtree Procedure

function varargout = QuadTree(varargin)

% Author: Ashok T Gaikwad

% Purpose: Region Oriented Segmentation

gui_Singleton = 1;

gui_State = struct('gui_Name',  mfilename, ...
   'gui_Singleton',  gui_Singleton, ...
   'gui_OpeningFcn', @QuadTree_OpeningFcn, ...
   'gui_OutputFcn',  @QuadTree_OutputFcn, ...
   'gui_LayoutFcn',  [] , ...
   'gui_Callback',   []);

if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{1});
else
    gui_mainfcn(gui_State, varargin{1});
end

% End initialization code - DO NOT EDIT
% --- Executes just before QuadTree is made visible.

function QuadTree_OpeningFcn(hObject, eventdata, handles, varargin)
    % This function has no output args, see OutputFcn.
    % hObject    handle to figure
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    structure with handles and user data (see GUIDATA)
    % varargin   command line arguments to QuadTree (see VARARGIN)

    % Choose default command line output for QuadTree
    handles.output = hObject;

    % Update handles structure
    guidata(hObject, handles);

    % UIWAIT makes QuadTree wait for user response (see UIRESUME)
    % uiwait(handles.fig);

% --- Outputs from this function are returned to the command line.

function varargout = QuadTree_OutputFcn(hObject, eventdata, handles)
    % varargout  cell array for returning output args (see VARARGOUT);
    % hObject    handle to figure
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

\[\textit{function img\_Callback(hObject, eventdata, handles)}\]

% hObject handle to img (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of img as text
% str2double(get(hObject,'String')) returns contents of img as a double

% --- Executes during object creation, after setting all properties.

\[\textit{function img\_CreateFcn(hObject, eventdata, handles)}\]

% hObject handle to img (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns called
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'))
;
end

function getImage_Callback(hObject, eventdata, handles)

    x=char(get(handles.img,'String'))
    i=imread(x);
    imshow(i);
    axes(handles.axes1);
    axes(handles.axes2);
imhist(i);

% --- Executes on button press in ImgQuadTree.

function ImgQuadTree_Callback(hObject, eventdata, handles)

% hObject    handle to ImgQuadTree (see GCBO)

% eventdata  reserved - to be defined in a future version of MATLAB

% handles    structure with handles and user data (see GUIDATA)

nb=0;

x=char(get(handles.img,'String'))
i=imread(x);
sz=size(i);

%Set the Original Size
set(handles.text21,'String',sz(1,1));
set(handles.text23,'String',sz(1,2));
asz=size(sz);

% Auto change the Type to Gray
if asz(1,2)==3
    j=rgb2gray(i);
i=j;
end

% Make the Image Square
if sz(1,1)<sz(1,2)
disp=sz(1,2)-sz(1,1)
i(sz(1,1)+disp,sz(1,2))=0;

else
    disp=sz(1,1)-sz(1,2)
    i(sz(1,1),sz(1,2)+disp)=0;
end

% Auto Make the Image in 2 Power

sz=size(i);

% 256
if sz(1,1)<256
    disp=256-sz(1,1)
    i(sz(1,1)+disp,sz(1,2)+disp)=0;
end

% 512
if (sz(1,1)>256 && sz(1,1)<512)
    disp=512-sz(1,1)
    i(sz(1,1)+disp,sz(1,2)+disp)=0;
end

% 1024
if (sz(1,1)>512 && sz(1,1)<1024)
    disp=1024-sz(1,1)
    i(sz(1,1)+disp,sz(1,2)+disp)=0;
end
if (sz(1,1)>1024 && sz(1,1)<2048)
    disp=2048-sz(1,1)
    i(sz(1,1)+disp,sz(1,2)+disp)=0;
end
if (sz(1,1)>2048 && sz(1,1)<4096)
    disp=4096-sz(1,1)
    i(sz(1,1)+disp,sz(1,2)+disp)=0;
end

sz=size(i);
sz;

disp=0;
sz=size(i);
set(handles.szX,'String',sz(1,1));
set(handles.szY,'String',sz(1,2));

thr= get(handles.popup2,'Value');

%Determine the Threshold Define
if thr>0
    switch thr
    case 1
        thrval=0.05
case 2
  thrval=0.10

case 3
  thrval=0.15

case 4
  thrval=0.20

case 5
  thrval=0.25

case 6
  thrval=0.30

case 7
  thrval=0.35

case 8
  thrval=0.40

case 9
  thrval=0.45

case 10
  thrval=0.50

case 11
  thrval=0.55

case 12
  thrval=0.60

case 13
thrval=0.65

case 14
    thrval=0.70

case 15
    thrval=0.75

case 16
    thrval=0.80

case 17
    thrval=0.85

case 18
    thrval=0.90

case 19
    thrval=0.95

case 20
    thrval=1.00
end
end

% Determine the No of Regions to Be Formed
reg=get(handles.popup1,'Value');

switch reg
    case 1
        nb=0;

139
S = qtdecomp(i,thrval);
blocks = repmat(uint8(0),size(S));
for dim = [1];
numblocks = length(find(S==dim));
nb=(nb+numblocks)-1;
if (numblocks > 0)
    values = repmat(uint8(1),[dim dim numblocks]);
    values(2:dim,2:dim,:) = 0;
    blocks = qtsetblk(blocks,S,dim,values);
end
end
set(handles.text28,'String',nb-numblocks)
set(handles.text30,'String',numblocks)
set(handles.text25,'String',nb)
case 2
nb=0;
S = qtdecomp(i,thrval);
blocks = repmat(uint8(0),size(S));
for dim = [2 1];
numblocks = length(find(S==dim));
nb=(nb+numblocks)-1;
if (numblocks > 0)
    values = repmat(uint8(1),[dim dim numblocks]);
values(2:dim,2:dim,:) = 0;
blocks = qtsetblk(blocks,S,dim,values);
end
end

set(handles.text28,'String',nb-numblocks)
set(handles.text30,'String',numblocks)
%set(handles.text25,'String',2*1);
set(handles.text25,'String',nb);
case 3
%set(handles.text25,'String',2*2);
% validreg=2*2*2;
nb=0;
S = qtdecomp(i,thrval);
blocks = repmat(uint8(0),size(S));
for dim = [4 2 1];
numblocks = length(find(S==dim));
nb=(nb+numblocks)-1;
if (numblocks > 0)
values = repmat(uint8(1),[dim dim numblocks]);
values(2:dim,2:dim,:) = 0;
blocks = qtsetblk(blocks,S,dim,values);
end
end

set(handles.text28,'String',nb-numblocks)
set(handles.text30,'String',numblocks)
set(handles.text25,'String',nb);

case 4

nb=0;

S = qtdecomp(i,thrval);
blocks = repmat(uint8(0),size(S));
for dim = [8 4 2 1];
numblocks = length(find(S==dim));
nb=(nb+numblocks)-1;
if (numblocks > 0)
values = repmat(uint8(1),[dim dim numblocks]);
values(2:dim,2:dim,:) = 0;
blocks = qtsetblk(blocks,S,dim,values);
end
end

set(handles.text28,'String',nb-numblocks)
set(handles.text30,'String',numblocks)
```matlab
set(handles.text25,'String',nb);

case 5

%set(handles.text25,'String',2*2*2*2);

nb=0;

S = qtdecomp(i,thrval);

blocks = repmat(uint8(0),size(S));

for dim = [16 8 4 2 1];
    numblocks = length(find(S==dim));
    nb=(nb+numblocks)-1;
    if (numblocks > 0)
        values = repmat(uint8(1),[dim dim numblocks]);
        values(2:dim,2:dim,:) = 0;
        blocks = qtsetblk(blocks,S,dim,values);
    end
end

set(handles.text28,'String',nb-numblocks)
set(handles.text30,'String',numblocks)
set(handles.text25,'String',nb);

case 6

set(handles.text25,'String',2*2*2*2*2);

nb=0;

S = qtdecomp(i,thrval);
```
blocks = repmat(uint8(0),size(S));
for dim = [32 16 8 4 2 1];
    numblocks = length(find(S==dim));
    nb=(nb+numblocks)-1;
    if (numblocks > 0)
        values = repmat(uint8(1),[dim dim numblocks]);
        values(2:dim,2:dim,:) = 0;
        blocks = qtsetblk(blocks,S,dim,values);
    end
end

set(handles.text28,'String',nb-numblocks)
set(handles.text30,'String',numblocks)
set(handles.text25,'String',nb)
case 7
    nb=0;
set(handles.text25,'String',2*2*2*2*2*2);
S = qtdecomp(i,thrval);
    blocks = repmat(uint8(0),size(S));
for dim = [64 32 16 8 4 2 1];
    numblocks = length(find(S==dim));
    nb=(nb+numblocks)-1;
    if (numblocks > 0)
values = repmat(uint8(1),[dim dim numbblocks]);
values(2:dim,2:dim,:) = 0;
blocks = qtsetblk(blocks,S,dim,values);
end
end

set(handles.text28,'String',nb-numblocks)
set(handles.text30,'String',numblocks)
set(handles.text25,'String',nb)
case 8
nb=0;
set(handles.text25,'String',2*2*2*2*2*2*2);
S = qtdecomp(i,thrval);
blocks = repmat(uint8(0),size(S));
for dim = [128 64 32 16 8 4 2 1];
numblocks = length(find(S==dim));
nb=(nb+numblocks)-1;
if (numblocks > 0)
values = repmat(uint8(1),[dim dim numbblocks]);
values(2:dim,2:dim,:) = 0;
blocks = qtsetblk(blocks,S,dim,values);
end
end
end
case 9

nb=0;

set(handles.text25,'String',2*2*2*2*2*2*2);

S = qtdecomp(i,thrval);

blocks = repmat(uint8(0),size(S));

for dim = [256 128 64 32 16 8 4 2 1];
    numbblocks = length(find(S==dim));
    nb=(nb+numblocks)-1;
    if (numblocks > 0)
        values = repmat(uint8(1),[dim dim numbblocks]);
        values(2:dim,2:dim,:) = 0;
        blocks = qtsetblk(blocks,S,dim,values);
    end
end

case 10

nb=0;

set(handles.text25,'String',2*2*2*2*2*2*2*2*2*2);

S = qtdecomp(i,thrval);
blocks = repmat(uint8(0),size(S));
for dim = [512 256 128 64 32 16 8 4 2 1];
    numblocs = length(find(S==dim));
    nb=(nb+numblocks)-1;
    if (numblocks > 0)
        values = repmat(uint8(1),[dim dim numblocks]);
        values(2:dim,2:dim,:) = 0;
        blocks = qtsetblk(blocks,S,dim,values);
    end
end

set(handles.text28,'String',nb-numblocks)
set(handles.text30,'String',numblocks)
set(handles.text25,'String',nb)
end
blocks(end,1:end) =1;
blocks(1:end,end) = 1;
figure
subplot(2,2,1),imshow(blocks,[])
subplot(2,2,2),imhist(blocks)
%subplot(2,2,3),imagesc(i)
%figure(figure1);
function popup2_CreateFcn(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.

% hObject    handle to popup2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.

if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'))
end

% --- Executes on selection change in popup1.
function popup1_Callback(hObject, eventdata, handles)

% hObject    handle to popup1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popup1 contents as cell array
% contents{get(hObject,'Value')} returns selected item from popup1

% --- Executes during object creation, after setting all properties.

function popup1_CreateFcn(hObject, eventdata, handles)

% hObject    handle to popup1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
% See ISPC and COMPUTER.

if ispc
    set(hObject,'BackgroundColor','white');
else

149
set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'))

;

end

% --- Executes on selection change in popup2.

function popup2_Callback(hObject, eventdata, handles)

% hObject    handle to popup2 (see GCBO)

% eventdata  reserved - to be defined in a future version of MATLAB

% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popup2 contents as cell
array

% contents{get(hObject,'Value')} returns selected item from popup2


function blocks = ComputeBlocks(S);

blocks = repmat(uint8(0), size(S));

for dim = [128 64 32 16 8 4 2 1];

numBlocks = length(find(S == dim));

if (numBlocks > 0)

values = repmat(uint8(1), [dim dim numBlocks]);

end
values(2:dim, 2:dim, :) = 0;
blocks = qtsetblk(blocks, S, dim, values);
end
end
blocks(end,1:end) = 1;
blocks(1:end,end) = 1;

%%%%
%%%% Subfunction ComputeMeans
%%%%

function means = ComputeMeans(I, S)

means = I;
for dim = [128 64 32 16 8 4 2 1];
values = qtgetblk(I, S, dim);
if (~isempty(values))
try
  % new functionality
  doublesum = sum(sum(values,1,'double'),2);
catch
  % old functionality
  doublesum = sum(sum(values,1),2);
end
means = qtsetblk(means, S, dim, doublesum ./ dim^2);
end
end
5.4.3 Watershed Segmentation Procedure

function varargout = WatershedEx(varargin)

% WATERSHEDEX M-file for WatershedEx.fig
% WATERSHEDEX, by itself, creates a new WATERSHEDEX or raises the existing
% singleton*.
% H = WATERSHEDEX returns the handle to a new WATERSHEDEX or the handle to
% the existing singleton*.
% WATERSHEDEX('CALLBACK',hObject,eventData,handles,...) calls the local
% function named CALLBACK in WATERSHEDEX.M with the given input arguments.
% WATERSHEDEX('Property','Value',...) creates a new WATERSHEDEX or raises the
% existing singleton*. Starting from the left, property value pairs are applied to the GUI before WatershedEx_OpeningFunction gets called. An
% unrecognized property name or invalid value makes property application
% stop. All inputs are passed to WatershedEx_OpeningFcn via varargin.
%
% *See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one
% instance to run (singleton)".
% See also: GUIDE, GUIDATA, GUIHANDLES
% Copyright 2002-2003 The MathWorks, Inc.
% Edit the above text to modify the response to help WatershedEx
% Last Modified by GUIDE v2.5 23-Mar-2002 16:11:12
% Begin initialization code - DO NOT EDIT

gui_Singleton = 1;

gui_State = struct('gui_Name', mfilename, ...
                    'gui_Singleton', gui_Singleton, ...
                    'gui_OpeningFcn', @WatershedEx_OpeningFcn, ...
                    'gui_OutputFcn', @WatershedEx_OutputFcn, ...
                    'gui_LAYOUTFcn', [], ...
                    'gui_Callback', []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
    [varargout{1:nargout}] = guimainfn(guistate, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end

% End initialization code - DO NOT EDIT

% --- Executes just before WatershedEx is made visible.

function WatershedEx_OpeningFcn(hObject, eventdata, handles, varargin)

% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to WatershedEx (see VARARGIN)
% Choose default command line output for WatershedEx
handles.output = hObject;
% Update handles structure
guidata(hObject, handles);
% UIWAIT makes WatershedEx wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.

function varargout = WatershedEx_OutputFcn(hObject, eventdata, handles)

% varargout cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1 as a double

% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'))
;
end

% --- Executes on button press in pushbutton1.

function pushbutton1_Callback(hObject, eventdata, handles)

  % hObject    handle to pushbutton1 (see GCBO)
  % eventdata  reserved - to be defined in a future version of MATLAB
  % handles    structure with handles and user data (see GUIDATA)

% --- Executes on selection change in popupmenu1.

function popupmenu1_Callback(hObject, eventdata, handles)

  % hObject    handle to popupmenu1 (see GCBO)
  % eventdata  reserved - to be defined in a future version of MATLAB
  % handles    structure with handles and user data (see GUIDATA)
% Hints: contents = get(hObject,'String') returns popupmenu1 contents as cell array
% contents{get(hObject,'Value')} returns selected item from popupmenu1

% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)
    % hObject    handle to popupmenu1 (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    empty - handles not created until after all CreateFcns called

    % Hint: popupmenu controls usually have a white background on Windows.
    % See ISPC and COMPUTER.
    if ispc
        set(hObject,'BackgroundColor','white');
    else

        set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'))
    ;
    end

% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)

    % hObject    handle to pushbutton2 (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    structure with handles and user data (see GUIDATA)
    x=char(get(handles.edit1,'String'))

    thr= get(handles.popupmenu1,'Value');

    switch thr
    case 1
        h=fspecial('average');
    case 2
        h=fspecial('disk');
    case 3

h = fspecial('gaussian');
case 4
    h = fspecial('laplacian');
case 5
    h = fspecial('log');
case 6
    h = fspecial('motion');
case 7
    h = fspecial('prewitt');
case 8
    h = fspecial('sobel');
case 9
    h = fspecial('unsharp');
case 10
    \%h = fspecial('test');
    h = [-2 -1 -2; 0 0 0; 2 1 2];
end
\%set(handles.text5, 'String', h);
i = imread(x);
fd = double(i);
g = sqrt(imfilter(fd, h, 'replicate') .^ 2);
l = watershed(g);
imshow(l);
g2=imclose(imopen(g,ones(3,3)),ones(3,3));
l1=watershed(g2);
imshow(l1);
wr2=l1==0;
f2=i;
f2(wr2)=255;
axes(handles.axes1)
imshow(i);
axes(handles.axes2)
imshow(l);
axes(handles.axes3)
imshow(f2);
5.5 System Design View (Main Menu)
5.6 Edge Detection Menu