**Q1)** %Daniel Peterson - 109091561

%6 shapes (3 rectanges, 1 circle, 1 square, and one ellipse) are created

%and placed on an image. The image has a 128 pixel value background so it

%is exactly in the middle of the intensity spectrum. Some of the images are

%brighter than the background and some are darker. Additive gaussian noise

%and impulse noise is also added to the image in 3 discrete intervals. For

%additive gaussian noise, the first variance is based on the image and is

%about 0.06. The following two variances are 0.1 and 0.2, which results in

%a grainier image as the variance increases. The impulse noise is applied

%with a random value with a given intensity. The initial intensity is 0.2,

%then 0.4 and then down to 0.05. The higher the intensity, the grainier the

%image becomes.

I = imread('cameraman.tif');

img = I(:,:,1);

image\_thresholded = img;

image\_thresholded(img>3) = 128; %make background gray

fh = figure;

imshow(image\_thresholded, []);

hold on;

%make shapes

rectangle('Position', [50 70 30 60], 'FaceColor', [1 1 1]);%1800

rectangle('Position', [30 80 10 70], 'FaceColor', [0.8 0.8 0.8]);%700

rectangle('Position', [100 100 100 100], 'FaceColor', [0.2 0.2 0.2], 'Curvature', [1 1]);%31,415

rectangle('Position', [40 180 50 50], 'FaceColor', [0.4 0.4 0.4]); %2500

rectangle('Position', [20 20 200 10], 'FaceColor', [0.6 0.6 0.6], 'Curvature', [1 1]); %6283

rectangle('Position', [90 45 110 10], 'FaceColor', [0.9 0.9 0.9]); %1100

frm = getframe(fh);

%imwrite(image\_thresholded, 'SEG1.tif', 'tif');

imwrite(frm.cdata, 'SEG1.tif');

imagearea = frm.cdata ~= 128;

%Calculate measurements for the image

areas = regionprops(imagearea, 'Area');

%add gaussian noise to the image

[myimage, map] = imread('SEG1.tif');

myimage = double(myimage) / 255;

v = var(myimage(:));

mean = 0;

Igauss = imnoise(myimage, 'gaussian', mean, v);

v = 0.1;

Igauss2 = imnoise(myimage, 'gaussian', mean, v);

v = 0.2;

Igauss3 = imnoise(myimage, 'gaussian', mean, v);

%add impulse noise to the image

randomimage = im2double(imread('SEG1.tif'));

p = 0.2;

Irand = (randomimage + p\*rand(size(randomimage)))/(1+p);

p = 0.4;

Irand2 = (randomimage + p\*rand(size(randomimage)))/(1+p);

p = 0.05;

Irand3 = (randomimage + p\*rand(size(randomimage)))/(1+p);

figure;

imshow([randomimage Irand Igauss Irand2 Igauss2 Irand3 Igauss3]);

%save all images

imwrite(Irand, 'SEG3a.tif');

imwrite(Irand2, 'SEG3b.tif');

imwrite(Irand3, 'SEG3c.tif');

imwrite(Igauss, 'SEG2a.tif');

imwrite(Igauss2, 'SEG2b.tif');

imwrite(Igauss3, 'SEG2c.tif');

**Q2)** %Daniel Peterson - 109091561

%This calculates the area error and labeling error for the original image

%The true (based on input length and width) area of the images is provided

%and the area calculated by matlab is calculated. The two results are then

%compared and the error is determined

img = imread('SEG1.tif');

b = img(:,:,1);

true\_area1 = 1800;

shape1 = b > 240;

area1 = bwarea(shape1); %1800

true\_area2 = 700;

shape2 = b == 230;

area2 = bwarea(shape2); %700

true\_area3 = 31415;

shape3 = b == 204;

area3 = bwarea(shape3); %31,415

true\_area4 = 2500;

shape4 = b == 153;

area4 = bwarea(shape4); %2500

true\_area5 = 6283;

shape5 = b == 102;

area5 = bwarea(shape5); %6283

true\_area6 = 1100;

shape6 = b == 51;

area6 = bwarea(shape6); %1100

true\_sum = true\_area1 + true\_area2 + true\_area3 + true\_area4 + true\_area5 + true\_area6;

area\_error = (1-((true\_sum - area1)/true\_sum));

area\_error2 = (1-((true\_sum - area2)/true\_sum));

area\_error3 = (1-((true\_sum - area3)/true\_sum));

area\_error4 = (1-((true\_sum - area4)/true\_sum));

area\_error5 = (1-((true\_sum - area5)/true\_sum));

area\_error6 = (1-((true\_sum - area6)/true\_sum));

area\_error\_percentage = 100 \* area\_error;

area\_error2\_percentage = 100 \* area\_error2;

area\_error3\_percentage = 100 \* area\_error3;

area\_error4\_percentage = 100 \* area\_error4;

area\_error5\_percentage = 100 \* area\_error5;

area\_error6\_percentage = 100 \* area\_error6;

total\_area\_error = 100\*(area\_error + area\_error2 + area\_error3 + area\_error4 + area\_error5 + area\_error6);

label\_error = 100 \* abs(true\_area1-area1)/true\_area1;

label\_error2 = 100 \* abs(true\_area2-area2)/true\_area2;

label\_error3 = 100 \* abs(true\_area3-area3)/true\_area3;

label\_error4 = 100 \* abs(true\_area4-area4)/true\_area4;

label\_error5 = 100 \* abs(true\_area5-area5)/true\_area5;

label\_error6 = 100 \* abs(true\_area6-area6)/true\_area6;

**Q3)**

%Daniel Peterson - 109091561

%The following segments each image by basic segmentation and chan-vese

%segmentation. Area\_error# represents the error in area for shape#, the

%percentage is represented by the respective name as well. True\_area#

%represents the true area of shape# based on the object's parameters upon

%creation. Label\_error# represents the labeling error for shape# as a

%percentage. Total\_area\_error represents the total error for all objects as

%a percentage

img = imread('SEG1.tif');

b = img(:,:,1);

calcError(b); %thresholding

%chan-vese

mask = zeros(size(img,1),size(img,2));

mask(10:150,140:250) = 1;

region\_seg(I, m, 200); %-- Run segmentation for 200 iterations

img = imread('SEG2a.tif');

b = img(:,:,1);

calcError(b);%thresholding

%chan-vese

mask = zeros(size(img,1),size(img,2));

mask(10:150,140:250) = 1;

region\_seg(I, m, 200); %-- Run segmentation for 200 iterations

img = imread('SEG2b.tif');

b = img(:,:,1);

calcError(b);%thresholding

%chan-vese

mask = zeros(size(img,1),size(img,2));

mask(10:150,140:250) = 1;

region\_seg(I, m, 200); %-- Run segmentation for 200 iterations

img = imread('SEG2c.tif');

b = img(:,:,1);

calcError(b);%thresholding

%chan-vese

mask = zeros(size(img,1),size(img,2));

mask(10:150,140:250) = 1;

region\_seg(I, m, 200); %-- Run segmentation for 200 iterations

img = imread('SEG3a.tif');

b = img(:,:,1);

calcError(b);%thresholding

%chan-vese

mask = zeros(size(img,1),size(img,2));

mask(10:150,140:250) = 1;

region\_seg(I, m, 200); %-- Run segmentation for 200 iterations

img = imread('SEG3b.tif');

b = img(:,:,1);

calcError(b);%thresholding

%chan-vese

mask = zeros(size(img,1),size(img,2));

mask(10:150,140:250) = 1;

region\_seg(I, m, 200); %-- Run segmentation for 200 iterations

img = imread('SEG3c.tif');

b = img(:,:,1);

calcError(b);%thresholding

%chan-vese

mask = zeros(size(img,1),size(img,2));

mask(10:150,140:250) = 1;

region\_seg(I, m, 200); %-- Run segmentation for 200 iterations

%The following code was found online and implements chan-vese segmentation

%The images are read and a mask is applied to differentiate between the

%foreground and background. The initial area expands and contours around

%edges when it finds other objects

function region\_seg(img,mask,max\_iterations,alpha,display)

%-- default value for parameter alpha is .1

if(~exist('alpha','var'))

alpha = .2;

end

%-- default behavior is to display intermediate outputs

if(~exist('display','var'))

display = true;

end

I = im2graydouble(img);

phi = bwdist(mask)-bwdist(1-mask)+im2double(mask)-.5;

%--main loop

for iterations = 1:max\_iterations % Note: no automatic convergence test

idx = find(phi <= 1.2 & phi >= -1.2); %get the curve's narrow band

%-- find interior and exterior mean

upts = find(phi<=0); % interior points

vpts = find(phi>0); % exterior points

u = sum(I(upts))/(length(upts)+eps); % interior mean

v = sum(I(vpts))/(length(vpts)+eps); % exterior mean

F = (I(idx)-u).^2-(I(idx)-v).^2; % force from image information

curvature = get\_curvature(phi,idx); % force from curvature penalty

dphidt = F./max(abs(F)) + alpha\*curvature; % gradient descent to minimize energy

%-- maintain the CFL condition

dt = .45/(max(dphidt)+eps);

%-- evolve the curve

phi(idx) = phi(idx) + dt.\*dphidt;

%-- Keep SDF smooth

phi = sussman(phi, .5);

end

segmented\_image = phi<=0;

figure;

imshow(img);

title('Original Image');

figure;

imshow(mask);

title('Initial Mask');

figure;

imshow(segmented\_image);

title('Final Chan-Vese');

true\_area1 = 1800;

true\_area2 = 700;

true\_area3 = 31415;

true\_area4 = 2500;

true\_area5 = 6283;

true\_area6 = 1100;

true\_sum = true\_area1 + true\_area2 + true\_area3 + true\_area4 + true\_area5 + true\_area6;

areaAll = bwarea(segmented\_image);

label\_error\_all = 100 \* abs(true\_sum-areaAll)/true\_sum;

end

function curvature = get\_curvature(phi,idx)

[dimy, dimx] = size(phi);

[y x] = ind2sub([dimy,dimx],idx); % get subscripts

%-- get subscripts of neighbors

ym1 = y-1; xm1 = x-1; yp1 = y+1; xp1 = x+1;

%-- bounds checking

ym1(ym1<1) = 1; xm1(xm1<1) = 1;

yp1(yp1>dimy)=dimy; xp1(xp1>dimx) = dimx;

%-- get indexes for 8 neighbors

idup = sub2ind(size(phi),yp1,x);

iddn = sub2ind(size(phi),ym1,x);

idlt = sub2ind(size(phi),y,xm1);

idrt = sub2ind(size(phi),y,xp1);

idul = sub2ind(size(phi),yp1,xm1);

idur = sub2ind(size(phi),yp1,xp1);

iddl = sub2ind(size(phi),ym1,xm1);

iddr = sub2ind(size(phi),ym1,xp1);

%-- get central derivatives of SDF at x,y

phi\_x = -phi(idlt)+phi(idrt);

phi\_y = -phi(iddn)+phi(idup);

phi\_xx = phi(idlt)-2\*phi(idx)+phi(idrt);

phi\_yy = phi(iddn)-2\*phi(idx)+phi(idup);

phi\_xy = -0.25\*phi(iddl)-0.25\*phi(idur)...

+0.25\*phi(iddr)+0.25\*phi(idul);

phi\_x2 = phi\_x.^2;

phi\_y2 = phi\_y.^2;

%-- compute curvature

curvature = ((phi\_x2.\*phi\_yy + phi\_y2.\*phi\_xx - 2\*phi\_x.\*phi\_y.\*phi\_xy)./...

(phi\_x2 + phi\_y2 +eps).^(3/2)).\*(phi\_x2 + phi\_y2).^(1/2);

end

%-- Converts image to one channel (grayscale) double

function img = im2graydouble(img)

[dimy, dimx, c] = size(img);

if(isfloat(img)) % image is a double

if(c==3)

img = rgb2gray(uint8(img));

end

else % image is a int

if(c==3)

img = rgb2gray(img);

end

img = double(img);

end

end

%-- level set re-initialization by the sussman method

function D = sussman(D, dt)

% forward/backward differences

a = D - shiftR(D); % backward

b = shiftL(D) - D; % forward

c = D - shiftD(D); % backward

d = shiftU(D) - D; % forward

a\_p = a; a\_n = a; % a+ and a-

b\_p = b; b\_n = b;

c\_p = c; c\_n = c;

d\_p = d; d\_n = d;

a\_p(a < 0) = 0;

a\_n(a > 0) = 0;

b\_p(b < 0) = 0;

b\_n(b > 0) = 0;

c\_p(c < 0) = 0;

c\_n(c > 0) = 0;

d\_p(d < 0) = 0;

d\_n(d > 0) = 0;

dD = zeros(size(D));

D\_neg\_ind = find(D < 0);

D\_pos\_ind = find(D > 0);

dD(D\_pos\_ind) = sqrt(max(a\_p(D\_pos\_ind).^2, b\_n(D\_pos\_ind).^2) ...

+ max(c\_p(D\_pos\_ind).^2, d\_n(D\_pos\_ind).^2)) - 1;

dD(D\_neg\_ind) = sqrt(max(a\_n(D\_neg\_ind).^2, b\_p(D\_neg\_ind).^2) ...

+ max(c\_n(D\_neg\_ind).^2, d\_p(D\_neg\_ind).^2)) - 1;

D = D - dt .\* D ./ sqrt(D.^2 + 1) .\* dD;

end

function shift = shiftD(M)

shift = shiftR(M')';

end

function shift = shiftL(M)

shift = [ M(:,2:size(M,2)) M(:,size(M,2)) ];

end

function shift = shiftR(M)

shift = [ M(:,1) M(:,1:size(M,2)-1) ];

end

function shift = shiftU(M)

shift = shiftL(M')';

end

%Thresholds images and calculates area and labeling error

function x = calcError(b)

%BASIC THRESHOLDING

true\_area1 = 1800;

true\_area2 = 700;

true\_area3 = 31415;

true\_area4 = 2500;

true\_area5 = 6283;

true\_area6 = 1100;

%Segmentation of images by thresholding

threshold = 240;

shape1 = b > threshold;

area1 = bwarea(shape1); %1800

threshold = 230;

shape2 = (b == threshold);

area2 = bwarea(shape2); %700

threshold = 204;

shape3 = (b == threshold);

area3 = bwarea(shape3); %31,415

threshold = 153;

shape4 = (b == threshold);

area4 = bwarea(shape4); %2500

threshold = 102;

shape5 = (b == threshold);

area5 = bwarea(shape5); %6283

threshold = 51;

shape6 = (b == threshold);

area6 = bwarea(shape6); %1100

threshold = 128;

allshapes = (b ~= threshold);

areaAll = bwarea(allshapes);

%Area error

true\_sum = true\_area1 + true\_area2 + true\_area3 + true\_area4 + true\_area5 + true\_area6;

area\_error = (1-((true\_sum - area1)/true\_sum));

area\_error2 = (1-((true\_sum - area2)/true\_sum));

area\_error3 = (1-((true\_sum - area3)/true\_sum));

area\_error4 = (1-((true\_sum - area4)/true\_sum));

area\_error5 = (1-((true\_sum - area5)/true\_sum));

area\_error6 = (1-((true\_sum - area6)/true\_sum));

area\_error\_percentage = 100 \* area\_error;

area\_error2\_percentage = 100 \* area\_error2;

area\_error3\_percentage = 100 \* area\_error3;

area\_error4\_percentage = 100 \* area\_error4;

area\_error5\_percentage = 100 \* area\_error5;

area\_error6\_percentage = 100 \* area\_error6;

total\_area\_error = 100 \* (area\_error + area\_error2 + area\_error3 + area\_error4 + area\_error5 + area\_error6);

%Labeling error

label\_error = 100 \* abs(true\_area1-area1)/true\_area1;

label\_error2 = 100 \* abs(true\_area2-area2)/true\_area2;

label\_error3 = 100 \* abs(true\_area3-area3)/true\_area3;

label\_error4 = 100 \* abs(true\_area4-area4)/true\_area4;

label\_error5 = 100 \* abs(true\_area5-area5)/true\_area5;

label\_error6 = 100 \* abs(true\_area6-area6)/true\_area6;

%Segmented shapes from image

figure;

imshow(shape1);

title('Shape 1');

figure;

imshow(shape2);

title('Shape 2');

figure;

imshow(shape3);

title('Shape 3');

figure;

imshow(shape4);

title('Shape 4');

figure;

imshow(shape5);

title('Shape 5');

figure;

imshow(shape6);

title('Shape 6');

figure;

imshow(allshapes);

title('All Shapes');

end

**Q4)** %Daniel Peterson - 109091561

%The term watershed refers to the region between two areas that divides

%the two systems. In a way, this is similar to an edge in an image, which

%divides objects. The watershed segmentation method divides images by

%determining which pixels belong with which objects. The watershed method

%treats images as surfaces, where dark areas are low and bright areas are

%high. By applying the watershed method, the edges of the image, where the

%high and low areas meet, become the ridges and defines those areas

%load all images and call watershed function

img = imread('SEG1.tif');

b = img(:,:,1);

calcWatershed(b);

img = imread('SEG2a.tif');

b = img(:,:,1);

calcWatershed(b);

img = imread('SEG2b.tif');

b = img(:,:,1);

calcWatershed(b);

img = imread('SEG2c.tif');

b = img(:,:,1);

calcWatershed(b);

img = imread('SEG3a.tif');

b = img(:,:,1);

calcWatershed(b);

img = imread('SEG3b.tif');

b = img(:,:,1);

calcWatershed(b);

img = imread('SEG3c.tif');

b = img(:,:,1);

calcWatershed(b);

%calculates and displays the watershed transform

function x = calcWatershed(b)

%shows original image

figure

imshow(b, [], 'InitialMagnification', 'fit');

title('Original B/W Image')

%removes background and creates distance transform

threshold = 128; %background color

b = (b ~= threshold);%get all objects not in background

D = bwdist(~b);

figure

imshow(D,[],'InitialMagnification','fit')

title('Distance transform of image')

%calculates the watershed of the distance transform and displays

D = -D;

D(~b) = Inf; %remove background pixels

L = watershed(D);

L(~b) = 0; %label background as 0

rgb = label2rgb(L,'jet',[.5 .5 .5]);

figure

imshow(rgb,'InitialMagnification','fit')

title('Watershed transform of image')

end

**Q5)** %Daniel Peterson - 109091561

%I'm not really sure how do approach this problem so I attempted to load

%the image and discretize the heat equations for the x and y domains

%Three iterations take place, from 0 to 1 in 0.1 increments (10), from 0 to

%10 in 0.1 increments (100) and from 0 to 100 in 0.1 increments (1000) for

%the image

img = imread('heart.jpg');

h = img(:,:,1);

applyHeat10(h);

applyHeat100(h);

applyHeat1000(h);

function applyHeat10(h)

figure;

imshow(h);

title('Original Image');

dt = 0.1;

T = 1;

[m,n]=size(h);

for t = 0:dt:T

h\_xx = h(:,[2:n,n])-2\*h +h(:,[1,1:n-1]);

h\_yy = h([2:m,m],:) - 2\*h + h([1,1:m-1],:);

L = h\_xx + h\_yy;

h = h + dt\*L;

end

figure;

imshow(h)

title('Smoothed Image');

end

function applyHeat100(h)

figure;

imshow(h);

title('Original Image');

dt = 0.1;

T = 10;

[m,n]=size(h);

for t = 0:dt:T

h\_xx = h(:,[2:n,n])-2\*h +h(:,[1,1:n-1]);

h\_yy = h([2:m,m],:) - 2\*h + h([1,1:m-1],:);

L = h\_xx + h\_yy;

h = h + dt\*L;

end

figure;

imshow(h)

title('Smoothed Image');

end

function applyHeat1000(h)

figure;

imshow(h);

title('Original Image');

dt = 0.1;

T = 100;

[m,n]=size(h);

for t = 0:dt:T

h\_xx = h(:,[2:n,n])-2\*h +h(:,[1,1:n-1]);

h\_yy = h([2:m,m],:) - 2\*h + h([1,1:m-1],:);

L = h\_xx + h\_yy;

h = h + dt\*L;

end

figure;

imshow(h)

title('Smoothed Image');

end