Supporting functions

The following functions were used in actual image processing filter, they support   
  
  
binary\_matrix

function [bmat] = binary\_matrix(mat)

bmat=(mat == 0) | (mat == 255);

k\_approximate\_matrix

function [kmat]= k\_approximate\_matrix(tmat,t,i,j,k)

kmat=tmat( i+t-k:i+t+k , j+t-k:j+t+k);

t\_symmetric\_pad\_matrix

function [tmat] = t\_symmetric\_pad\_matrix(mat,t)

dim=size(mat);

tmat=zeros(dim+2\*t);

tmat( (t+1:t+dim(1)) , (t+1:t+dim(2)) )=mat;

dim=size(tmat);

tmat( 1:t , : )=flip(tmat( t+1:2\*t , : ),1);

tmat( dim(1)-t+1:dim(1) , : )=flip(tmat( dim(1)-2\*t+1:dim(1)-t , : ),1);

tmat( : , 1:t )=flip(tmat( : , t+1:2\*t ),2);

tmat( : , dim(2)-t+1:dim(2) )=flip(tmat( : , dim(2)-2\*t+1:dim(2)-t ),2);

Filters

BASIC MEAN FILTER

%basic\_mean\_filter

function [im]= basic\_mean\_filter(im)

im=t\_symmetric\_pad\_matrix(im,1);

dim=size(im);

for i=2:dim(1)-1

for j=2:dim(2)-1

if(im(i,j)==0 || im(i,j)==255)

mask=im( (i-1):(i+1) , (j-1):(j+1) );

element=mean([mask(1,:),mask(2,1),mask(2,3),mask(3,:)]);

im(i,j)=element;

end

end

end

im=im(2:dim(1)-1,2:dim(2)-1);

im=uint8(im);

BASIC MEDIAN FILTER

%basic\_median\_filter

function [im]=basic\_median\_filter(im)

im=t\_symmetric\_pad\_matrix(im,1);

dim=size(im);

for i=2:dim(1)-1

for j=2:dim(2)-1

if(im(i,j)==0 || im(i,j)==255)

mask=im( (i-1):(i+1) , (j-1):(j+1) );

im(i,j)=median(mask,"all");

end

end

end

im=im(2:dim(1)-1,2:dim(2)-1);

im=uint8(im);

ADAPTIVE MEDIAN FILTER

%adaptive\_median\_filter

function [nim] = adaptive\_median\_filter(nim,t)

dim=size(nim);

tmat=t\_symmetric\_pad\_matrix(nim,t);

for i=1:dim(1)

for j=1:dim(2)

for k=1:t

kmat=k\_approximate\_matrix(tmat,t,i,j,k);

mini=min(min(kmat));

maxi=max(max(kmat));

med=median(kmat,"all");

if (mini<med && med<maxi) && (nim(i,j)==mini || nim(i,j)==maxi)

nim(i,j)=med;

end

end

end

end

MODIFIED DECISION BASED UNSYMMETRIC TRIMMED MEDIAN FILTER

%modified\_decision\_based\_unsymmetric\_trimmed\_median\_filter

function [nim] = modified\_decision\_based\_unsymmetric\_trimmed\_median\_filter(nim)

tmat=t\_symmetric\_pad\_matrix(nim,1);

bmat=binary\_matrix(nim);

btmat=t\_symmetric\_pad\_matrix(bmat,1);

dim=size(nim);

for i=1:dim(1)

for j=1:dim(2)

if bmat(i,j)

bkmat=k\_approximate\_matrix(btmat,1,i,j,1);

kmat=k\_approximate\_matrix(tmat,1,i,j,1);

if all(bkmat,"all")

nim(i,j)=mean(kmat,"all");

else

for index=1:9

if (kmat(index)==255) || (kmat(index)==0)

kmat(index)=nan;

end

end

nim(i,j)=median(kmat,'all','omitnan');

end

end

end

end

ADAPTIVE RIESZ MEAN FILTER

%adaptive\_riesz\_mean\_filter

function [nim] = adaptive\_riesz\_mean\_filter(nim)

nim=double(nim);

dim=size(nim);

bmat=binary\_matrix(nim);

for t=5:-1:1

tmat=t\_symmetric\_pad\_matrix(nim,t);

btmat=t\_symmetric\_pad\_matrix(bmat,t);

for i=1:dim(1)

for j=1:dim(2)

if bmat(i,j)==1

for k=1:t

bkmat=k\_approximate\_matrix(btmat,t,i,j,k);

if ~all(bkmat,"all")

kmat=k\_approximate\_matrix(tmat,t,i,j,k);

nim(i,j) = riesz\_mean(kmat,k);

end

end

end

end

end

end

nim=uint8(nim);

The following function riesz\_mean is explicitly is used only in Adaptive riesz mean filter

function [rm] = riesz\_mean(mat,k)

dim=length(mat);

num=0;

den=0;

ps=pixel\_similarity(k);

for s=1:dim

for t=1:dim

if (mat(s,t)~=0)&&(mat(s,t)~=255)

num=num+ps(s,t)\*mat(s,t);

den=den+ps(s,t);

end

end

end

rm=num/den;

DIFFERENT ADAPTIVE MODIFIED RIESZ MEAN FILTER

%different\_adaptive\_modified\_riesz\_mean\_filter

function [nim] = different\_adaptive\_modified\_riesz\_mean\_filter(nim)

dim=size(nim);

bmat=binary\_matrix(nim);

for t=5:-1:1

tmat=t\_symmetric\_pad\_matrix(nim,t);

for i=1:dim(1)

for j=1:dim(2)

if bmat(i,j)

for k=1:t

kmat=k\_approximate\_matrix(tmat,t,i,j,k);

med=median(kmat,"all");

if (0<med && med<255) && (nim(i,j)==0 || nim(i,j)==255)

nim(i,j)=modified\_riesz\_mean(kmat,k);

break

end

end

end

end

end

end

ALPHA TRIMMED MEAN FILTER

%alpha\_trimmed\_mean\_filter

function [im]=alpha\_trimmed\_mean\_filter(im)

dim=size(im);

tmat=t\_symmetric\_pad\_matrix(im,2);

for i=3:dim(1)+2

for j=3:dim(2)+2

mat=Window(tmat,i,j,5);

res=trimmed\_mean\_calculator(mat,5);

res=res/21;

im(i-2,j-2)=res;

end

end

The following functions trimmed\_mean\_calculator and window are explicitly is used only in Alpha trimmed mean filter

function[sum]=trimmed\_mean\_calculator(mat,d)

matlen=length(mat);

uplim=matlen-floor(d/2);

lowlim=floor(d/2);

mean\_ans=0;

for i=lowlim:uplim

mean\_ans=mean\_ans+mat(i);

end

sum=mean\_ans;

function[res]= Window(mat,i,j,ws)

res=zeros(ws\*ws);

k=1;

rlim=(ws-1)/2;

for a=i-rlim:i+rlim

for b=j-rlim:j+rlim

res(k)=mat(a,b);

k=k+1;

end

end

GEOMETRIC MEAN FILTER

function [im]=geometric\_mean\_filter(im)

dim=size(im);

tmat=t\_symmetric\_pad\_matrix(im,1);

for i=1:dim(1)

for j=1:dim(2)

kmat=k\_approximate\_matrix(tmat,1,i,j,1);

res=1;

count=0;

for m=1:3

for n=1:3

if ~(kmat(m,n)==0)||(kmat(m,n)==255)

res=res\*kmat(m,n);

count=count+1;

end

end

end

if count

res=res^(1/count);

end

im(i,j)=res;

end

end

im=uint8(im);

MODIFIED MEDIAN FILTER

function [im]=modified\_median\_filter(im)

dim=size(im);

bmat=binary\_matrix(im);

tmat=t\_symmetric\_pad\_matrix(im,1);

btmat=t\_symmetric\_pad\_matrix(bmat,1);

for i=1:dim(1)

for j=1:dim(2)

if bmat(i,j)

bkmat=k\_approximate\_matrix(btmat,1,i,j,1);

if ~(all(bkmat,"all"))

kmat=k\_approximate\_matrix(tmat,1,i,j,1);

im(i,j)=median(kmat,"all");

end

end

end

end

RECURSIVE SPLINE INTER-POLATION FILTER

function[mat] = recur(mat)

bmat = binary\_matrix(mat);

sz = size(mat);

for i = 2:sz(1)-1

for j = 2:sz(2)-1

if(bmat(i,j))

mask\_chk=bmat( (i-1):(i+1) , (j-1):(j+1) );

mask = mat( (i-1):(i+1) , (j-1):(j+1) );

val=sum(sum(bmat));

if(val>3)

mat(i,j)=test\_case1(mask);

% bmat(i,j)=0;

else

mat(i,j)=test\_case2(mask,mask\_chk);

% bmat(i,j)=0;

end

end

end

end

The following functions test\_case1 and test\_case2 are explicitly is used only in recursive spline interpolation filter

function[val] = test\_case1(mat)

val=mean\_val(mat);

val=ceil(val);

function[val]=test\_case2(mask,mask\_chk)

count=-1;

x1=[];

y1=[];

for s=1:3

for t=1:3

count=count+1;

if~(mask\_chk(s,t))

x1 = [x1,count];

y1 = [y1,mask(s,t)];

end

end

end

count =0;

xl = length(x1);

for i = 1:xl

j = x1(i);

if j < 4

count = count +1;

end

end

val = spline(x1,y1,var);

val = ceil(val);

end

SECTOR ROTATIONAL FILTER

function [im] = sector\_rotational\_filter(im)

dim=size(im);

t=5;

tmat=padarray(im,[t t],"symmetric");

bmat=binary\_matrix(im);

for i=1:dim(1)

for j=1:dim(2)

if bmat(i,j)

for k=2:t

kmat=k\_approximate\_matrix(tmat,t,i,j,k);

res=sec\_rot\_win\_res(kmat);

if res==-1

if k==t

im(i,j)=mean(kmat(:));

else

continue

end

else

im(i,j)=res;

end

end

end

end

end

The following function sec\_rot\_win\_res is explicitly is used only in sector rotational filter

function [res]=sec\_rot\_win\_res(kmat)

function call7

if ~(kmat(2,5)==255 || kmat(2,5)==0)

diff=5\*kmat(2,5)-sum(kmat(1:3,4))-kmat(3,5);

if diff<mini

mini=diff;

res=median([kmat(2:3,5);kmat(1:3,4)]);

end

end

end

function call9

if ~(kmat(2,4)==255 || kmat(2,4)==0)

diff=9\*kmat(2,4)-sum(kmat(1:4,5))-sum(kmat(3:4,4))-sum(kmat(2:3,3));

if diff<mini

mini=diff;

res=median([kmat(1:4,5);kmat(2:4,4);kmat(2:3,3)]);

end

end

end

function call11

if ~(kmat(2,5)==255 || kmat(2,5)==0)

diff=11\*kmat(2,5)-sum(kmat(1:5,6))-sum(kmat(3:5,5))-sum(kmat(2:4,4))-kmat(3,3);

if diff<mini

mini=diff;

res=median([kmat(1:5,6);kmat(2:5,5);kmat(2:4,4);kmat(3,3)]);

end

end

end

function call13

if ~(kmat(2,6)==255 || kmat(2,6)==0)

diff=11\*kmat(2,6)-sum(kmat(1:6,7))-sum(kmat(3:6,6))-sum(kmat(2:5,5))-sum(kmat(3:4,4))-kmat(3,3);

if diff<mini

mini=diff;

res=median([kmat(1:6,7);kmat(2:6,6);kmat(2:5,5);kmat(3:4,4);kmat(3,3)]);

end

end

end

k=(length(kmat)-1)/2;

res=-1;

mini=1000;

switch k

% for 5 by 5

case 2

for i=1:4

if ~(kmat(2,3)==255 || kmat(2,3)==0)

diff=2\*kmat(2,3)-kmat(1,3)-kmat(2,4);

if diff<mini

mini=diff;

res=median([kmat(2,3),kmat(1,3),kmat(2,4)]);

end

end

if ~(kmat(2,4)==255 || kmat(2,4)==0)

diff=2\*kmat(2,4)-kmat(3,4)-kmat(3,5);

if diff<mini

mini=diff;

res=median([kmat(2,4),kmat(3,4),kmat(3,5)]);

end

end

kmat=flip(transpose(kmat),2);

end

% for 7 by 7

case 3

for i=1:4

call7

kmat=transpose(kmat);

call7

kmat=flip(kmat);

end

% for 9 by 9

case 4

for i=1:4

call9

kmat=transpose(kmat);

call9

kmat=flip(kmat);

end

%for 11 by 11

case 5

for i=1:4

call11

kmat=transpose(kmat);

call11

kmat=flip(kmat);

end

case 6

for i=1:4

call13

kmat=transpose(kmat);

call13

kmat=flip(kmat);

end

end

end

MAX FILTER

function[ni]= max\_filter(ni)

[a,b]=size(ni);

tmat=t\_symmetric\_pad\_matrix(ni,1);

for i=1:a

for j=1:b

kmat=k\_approximate\_matrix(tmat,1,i,j,1);

maxi=0;

for m=1:3

for n=1:3

if ~(kmat(m,n)==0 || kmat(m,n)==255) && kmat(m,n)>maxi

maxi=kmat(m,n);

end

end

end

ni(i,j)=maxi;

end

end

MIN FILTER

function[ni]= min\_filter(ni)

[a,b]=size(ni);

tmat=t\_symmetric\_pad\_matrix(ni,1);

for i=1:a

for j=1:b

kmat=k\_approximate\_matrix(tmat,1,i,j,1);

mini=255;

for m=1:3

for n=1:3

if ~(kmat(m,n)==0 || kmat(m,n)==255) && kmat(m,n)<mini

mini=kmat(m,n);

end

end

end

ni(i,j)=mini;

end

end