

**Advanced Operating systems**

Made by: Ahmed Mohamed Hussien

Masters ID: 10049

Submitted to: Dr.seif

Assignment name

**Assignment #1**

# Question 1

## a.1.Integral Image code

function [image\_ii\_norm,image\_ii] =integral\_image()

% the function takes the array of the normal image >> image

% return the integral image >> image\_ii

image=imread('cameraman\_noise.tif');

image=double(image); %increasing the range of the image from unit8 to double size to allow the calculation of integral image

image\_ii=zeros(size(image,1),size(image,2)); %intializing the array of the integral image

cumulative\_sum=zeros(size(image,1),size(image,2)); %initializing the cumlative sum array

%calculatin the cumulative row sum

cumulative\_sum(:,1)=image(:,1);

cumulative\_sum(1,:)=image(1,:);

double(cumulative\_sum);

double(image\_ii);

for i=1:size(image,1)

for j=2:size(image,2)

cumulative\_sum(i,j)=cumulative\_sum(i,j-1)+image(i,j);

end

end

%calculating the integral image for a given image

image\_ii(1,:)=cumulative\_sum(1,:);

for j=1:size(image,2)

for i=2:size(image,1)

image\_ii(i,j)=image\_ii(i-1,j)+cumulative\_sum(i,j);

end

end

%normalizing the integral image to a range from 0 to 255

image\_ii\_norm=zeros(size(image,1),size(image,2)); %intailizing the image\_ii\_norm array

%getting the normalized integrated image

for i=1:size(image,1)

for j=1:size(image,2)

image\_ii\_norm(i,j) =(image\_ii(i,j)/max(image\_ii(:)))\*255;

end

end

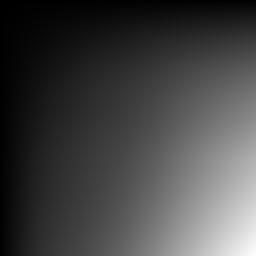
%displaying the integral\_image

image\_ii\_norm(:)=round(image\_ii\_norm(:)); % getting the integer values of the normalized values to display the image

image\_ii\_norm=uint8(image\_ii\_norm); % making the values of class uint8

imwrite(image\_ii\_norm,'camera\_Integ.jpg')

## a.2. Integral image picture



## b.1. Filter by integral image code

function[filtered\_image,integral\_image1] = avg\_filter(image,filter\_size)

%implementing the filter

image=double(image);

filtered\_image = zeros(size(image,1),size(image,2)); % initializing the filtered image array for preallocation

integral\_image1 = integral\_image(image); %creating the integral image of the input image

%filtering the image by getting the averag by using the integral image

for i=1:size(image,1)-filter\_size(1)

for j=1:size(image,2)-filter\_size(2)

filtered\_image(i+floor(filter\_size(1)/2),j+floor(filter\_size(2)/2)) = (1/(filter\_size(1)\*filter\_size(2)))\*(integral\_image1(i+filter\_size(1),j+filter\_size(2))+integral\_image1(i,j)-integral\_image1(i+filter\_size(1),j)-integral\_image1(i,j+filter\_size(2)));

end

end

%displaying the filtered\_image

filtered\_image(:)=round(filtered\_image(:)); % getting the integer values to display the image

filtered\_image=uint8(filtered\_image); % making the values of class uint8

imwrite(filtered\_image,'camera\_filt\_5.jpg')

## b.2. Filter by integral image output

3x3



5x5



## C

The integral image if calculated once it will be better than calculating the convolution of the image every time as it will save the time of the for loop of the convolution which will optimize the code and save more time

# Question 2

## Code

function [rotated\_image,croped\_image] = rotate( image, ceta)

% function take the image array + ceta and return the image rotated by ceta by two methods 1-image\_rotated(caluclated by linear interpoation) 2-rotated\_image\_nearest\_neighbour

% ceta will be in degree

% the program will return the full image by only for values of ceta between

% 0 and 90 degrees other values for ceta is not granted

image0=zeros(2\*size(image,1),2\*size(image,2));

rotated\_image=zeros(2\*size(image,1),2\*size(image,2)); %intializing the rotated\_image with zeros for preallocation

T=[cos((ceta/180)\*pi) -sin((ceta/180)\*pi);sin((ceta/180)\*pi) cos((ceta/180)\*pi)]; % transforamation function

%putting the image in a bigger image for rotation without croping any part

%of the image

for x=1:size(image,1)

for y=1:size(image,2)

image0(x+size(image,1)/4,y+size(image,2)/4)=image(x,y);

end

end

% linear image and nearest neighbour

for X\_new=1:2\*size(image,1)

for Y\_new=1:2\*size(image,2)

if ceta < 45 % this condition is for determining the offset of the image for angles between 0 and 90

X = (inv(T) \* [X\_new ; Y\_new])+[-3\*ceta;3\*ceta] ; %offsets are chosen by trial and error there's no special formula

elseif ceta >= 45 && ceta <= 90

X = (inv(T) \* [X\_new ; Y\_new])+[-2\*(ceta);4.5\*ceta] ; %offsets are chosen by trial and error there's no special formula

else

if X\_new ==1 && Y\_new == 1

disp('valid range is from 0 to 90 image will not be mostly displayed because the offset is not calculated right ')

X = (inv(T) \* [X\_new ; Y\_new])+[-2\*(ceta);4.5\*ceta] ;

else

X = (inv(T) \* [X\_new ; Y\_new])+[-2\*(ceta);4.5\*ceta] ;

end

end

% calculating the values used in the interpolation equation (l,k)

% (a,b)

l=floor(X(1));

k=floor(X(2));

l=uint16(l);

k=uint16(k);

a=X(1)-l;

b=X(2)-k;

if k > 0 && l > 0 && k<size(image0,1) && l< size(image0,2)

rotated\_image(X\_new , Y\_new) = (1-a)\*(1-b)\*image0(l,k) + a\*(1-b)\*image0(l+1,k) + b\*(1-a)\*image0(l,k+1) + a\*b\*image0(1+l,1+k); %linear interpolation equation

else

end

end

end

%changing type of rotating image

rotated\_image=uint8(rotated\_image);

%croping the rotated image ( linear interpolation )

%cutting the upper edge and lower edge

countx=1; % this counter is to count the new size of the x (no. of rows )

for x=1:2\*size(image,1)

for y=1:2\*size(image,2)

if( (rotated\_image(x,y)==0) ) %by passing the rows that contain 0 values only

continue;

else

croped\_image1(countx,:)=rotated\_image(x,:); %taking the rows that contain other values into a new array

countx=countx+1;

break;

end

end

end

%cutting the right and left side

county=1; % this counter is to count the new size of the y (no. of colums )

for y=1:2\*size(image,2)

for x=1:2\*size(image,1)

if( (rotated\_image(x,y)==0) ) %by passing the colums that contain 0 values only

continue;

else

croped\_image(:,county)=croped\_image1(:,y); %taking the colums that contain other values into a new array

county=county+1;

break;

end

end

end

## Image rotated by 40



## Image rotated by 70

