Homework1 – Image Fundamentals

Arya Araban

Abstract

Geometric Transformations as well as Quantization & Interpolation can be considered the ground fundamentals we should know in order to study the more advanced topics of computer vision. With the help of these fundamentals, we can do operations like rotation, stretching, downsampling and upsampling. In this report we'll be exploring these operations and their uses.

Report Info

Date: 1399/08/10

Keywords:
Affine
Geometric
Transformations
Interpolation
Quantization

1- Introduction

As part of the image processing fundamentals, Geomatric Transformations are functions in which we perform transformations on an image, resulting in an output image, which the distance ratio between any two pixels always staying the same. Quantization is the process of converting a continuous range of values into a finite range of discreet values. Finally, Interpolation is the process by which the number of pixels comprising an image is increased to allow printing enlargements that are of higher quality than the original image.

2- Technical Description

Problem 1.1.1:

Given two visually similar images and N(N>=3) feature points, we want to find a mapping function to be able to align the two images.

What we mean here by feature points, can also be known as "tie points" which are corresponding points between the two pictures whose locations are known precisely in the first and second images, given to us as (uk , vk) for the first image and (xk , yk) for the second, where k = 1,2, ... , N

Solution:

The objective is to find an affine mapping which helps us transform the first image geometrically to produce an output image that is aligned with the second image

With the help of a matrix affine transformations shown as the following:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

As seen above, the affine transformation has 6 parameters, and we need atleast 3

way, and then finally concatenate the two images next to each other.
What I did was find 4 tie points, and then scale the second image onto the first image with the help of the two equations below:

x=c1v+c2w+c3vw+c4
y=c5v+c6w+c7vw+c8

During the estimation phase, (v, w) and (x, y) are the coordinates of tie points in the input and reference images respectively.
After solving the 8 linear equation system, the values of c1 through c8 will be found, thus ending the estimation phrase.
Now all that's left is the actual scaling of the second image, which happens with the equations above by denoting (v,w) as the coordinates of a pixel of the original image, and the (x,y) being the value the current pixel should be mapped to.

Problem 1.1.3:
Implement rotation (with interpolation) on an image.

Solution:
We know that in order to implement rotation the matrix affinity formula looks like below for each pixels:

x = v.cos(theta) – w.sin(theta)
y = v.cos(theta) + w.sin(theta)

with (v,y) being the points of the original pixel, and (x,y) being it's new coordinates. However, this doesn't take into consideration that we want to rotate across the center. In order to do that we do the following before mapping the pixel coordinates:
v = v-centerX;
w = w-centerY;

pairs of corresponding points, where usually we use more than 3 pairs in order to find the best fitting affine parameters.
Let's suppose we have N pairs of corresponding points pi and pi',
then for i= 1,2,…N we'll have:

x'i = a11xi + a12yi +a13
y'I = a21xi +a22yi+ a23

Now we'll have two sets of Linear equations in the form of MA = B:

$$\begin{bmatrix} x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots \\ x_n & y_n & 1 \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \end{bmatrix} = \begin{bmatrix} x'_1 \\ \vdots \\ x'_n \end{bmatrix}$$

$$\begin{bmatrix} x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots \\ x_n & y_n & 1 \end{bmatrix} \begin{bmatrix} a_{21} \\ a_{22} \\ a_{23} \end{bmatrix} = \begin{bmatrix} y'_1 \\ \vdots \\ y'_n \end{bmatrix}$$

We can use the "Linear Least Squared" solution to solve this equation as follows:

$$\mathbf{a} = (\mathbf{M}^\top \mathbf{M})^{-1} \mathbf{M}^\top \mathbf{b}$$

now being able to find the unknown coefficents, we can plug in the values of each pixel of the second image in order to find the correct mapping of it to scale into that of the first image.

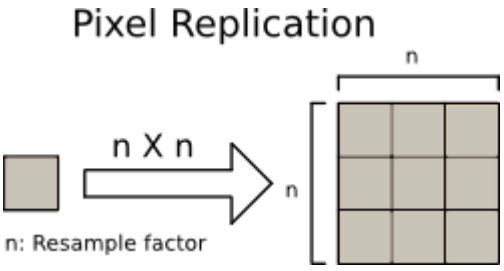
Problem 1.1.2:
Given two images of a moving car, taken in a very near timescale, we want to implement an algorithm that stiches the two images in order to make a "panaroma" image.

Solution:
In order to solve this problem, the appropriate approach to take is to first find a way to scale the second image to the first image, modify the second image in this

the one single average row or column.

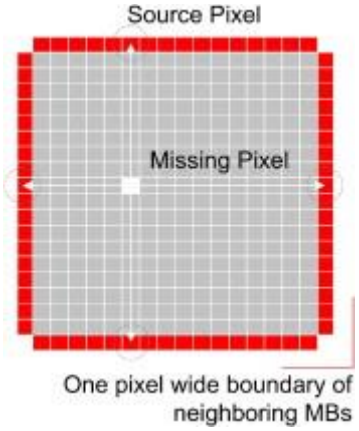
in order to reverse the effects of downsample we need to upsample. We'll explore two methods to do this.

METHOD 1 – PIXEL REPLICATION:



As the above image illustrates, pixel replication is an upsample method, which effects by increasing the number of pixels in an image, but without adding any data or detail.
here our resample factor is 2, so each pixel will take 4 pixel spaces of the original image.

METHOD 2 –
BILINEAR INTERPOLATION:



Bilinear interpolation replaces each missing pixel with a weighted average of the nearest pixels on the boundary of the 4 neighboring MBs as shown in Figure 9.5. The weights used are inversely

in order to do interpolation, we do what is known as "inverse mapping" that uses the nearest neighbour interpolation method.

Problem 1.2.1:
Do quantization on an image in (8, 16, 32, 64, 128) levels.

Solution:
Quantization is the process of converting a continuous range of values into a finite range of discreet values
If we consider the image in which we want to do quantization on having grayscale values in range [0…255] (256 possible different values), we can simply find

v = floor((Image_matrix) ./ (256/L))

where L is the number of desired Levels we want to quantize by.
As an example if we do this procedure for I = [255,23,10] and L = 2 it will give us [128,0,0]

Problem 1.2.2:
Downsample an image by the factor of 2, with and without an average filter, then upsample with pixel replication and bilinear interpolation

Solution:
Downsampling is the reduction in spatial Resolution(rows &columns) while keeping same two dimensional (2D) representation. It is typically used to reduce the storage and/or transmission requirements images. In order to downsample by a factor of two, We can either eliminate half of the samples (i.e eliminate rows &columns 1,3,5…),or we can take the average between each two rows or two columns and then replace them both, with

to work properly. The results were:

if zoomed in a bit on the picture, it is visible that there are black holes in the picture. After implementing the nearset neighbour interpolation (with, the help of inverse mapping), the black holes were gone. Here are the final results for rotating 30,45, and 80 degrees in order:

proportional to the distance of source and destination pixels. Note that for upsampling by a factor of 2, we consider half of the pixels as "missing", and then perform Bilinear Interpolation.

3- Describing the Results

Problem 1.1.2:

I implemented the explained algorithm, by first solving the Linear equations, and then by transforming the second image's pixels based on the newly found coefficients using two for loops, however there was a problem. These nested loops took a very long time to compute the locations of the new pixels, So I was forced to back down entirely from using this algorithm, and try another way to solve the problem.

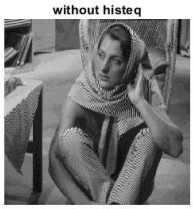
I considered solving this problem in a much more simpler manner, by just manually modifying both of the images,cropping a bit from both, and then finally stiching them together in order to make a panaroma. The results were almost close to perfect:

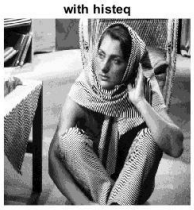
As seen above, the points seem to match pretty well, although there's a visible "separation line" between the two images.

Problem 1.1.3:

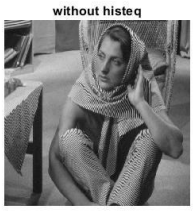
At first when implementing, I didn't manage to implement interpolation for rotation

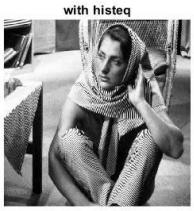
~16 levels~



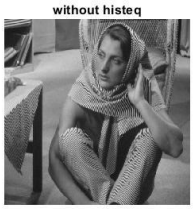


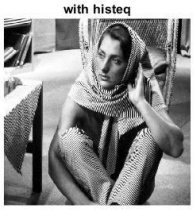
~32 levels~





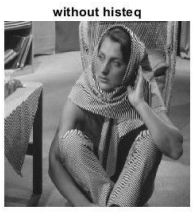
~64 layers~

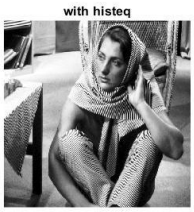


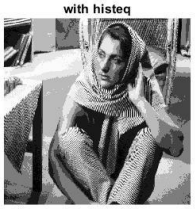


Problem 1.2.1:
I managed to implement the formula which gives quantization on any given level.
And then showing the quantized images, With and without using histeq side by side.
In my code I also found the MSE with the original image for both cases.
~8 levels~

~128 layers~



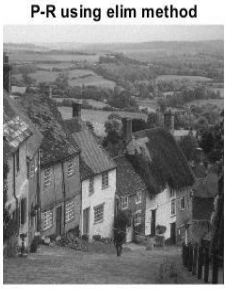




~METHOD2 – ELIMINATING ~

B-I using elim method

P-R using elim method



And the report for the MSE's between each of these and the original image:

Report MSE		
	Pixel Replication	Bilinear Interpolation
Averaging	133.42	65.23
Remove Row&Column	133.07	180.23

Which as seen above, for pixel replication it hardly matters weather we downsample with averaging or eliminating.
However for Bilinear Interpolation, averaging gives a drastically better result than removing rows&columns.

4- Appendix(code)

Problem 1.1.2:
The first set of code, that was unsuccessful, Due to the for loops taking a very long time to compute the output for each image:

And the MSE's:

Report MSE					
Level	8	16	32	64	128
Without Histeq	321	76	18	3	1
With Histeq	1136	1075	1027	1047	1086

Problem 1.2.2:
In order to downsample by a factor of 2, we either do the averaging method (which I implemented by reshaping the grayscale vector into a matrix, and then taking the average between two rows and two columns), or simply by removing half of the rows and columns.
I also implemented the Bilinear interpolation, and Pixel replication, each as separate functions, to be able to use them In the main program.
Now I'll be showing the results of each of the average and elimination methods, while showing the results of both upsample methods side-by-side
~METHOD1 – AVERAGING ~

B-I using avg method

P-R using avg method

Problem 1.2.1:

Quantization code, with comments:

```
in = imread('C:\Users\Arya\Desktop\computer vision\HW\Images\1\Barbara.bmp');
dataType = 'uint8'; %the original datatype
in = rgb2gray(in);
L = 64; %NUMBER OF LAYERS
figure(1);

[m,n] = size(in);

% we convert to double so we can do math on the original image
in = double(in);
%Doing the main computations for quantization
q = 256/L;
v = floor(in./q);
v = v.*q;

%return the images to their original datatype
v = uint8(v);
in = uint8(in);
%--- PLOT without histeq ----
subplot(1,2,1)
imshow(v);
title('without histeq')
without_histeq_mse = int64(immse(v,in))

%---NOW WITH HISTEQ ----
v = histeq(v);
with_histeq_mse = int64(immse(v,in))

subplot(1,2,2)
imshow(v);
title('with histeq')
```

Problem 1.2.2:

First I'll show how I implemented bilinear interpolation, along with comments:

```
function output = bilinear_interpolation(img)
 [X, Y] = size(img);
 output = zeros(2*X,2*Y);
 % If the zooming factor is 2, then the mapped pixel point in the original
 %image is given by 'r' and 'c' as r = floor(x/2) and c = floor(y/2)
for i = 1:2*X
    for j = 1: 2*Y
        x1 = uint32(floor(i/2));
        x2 = uint32(ceil(i/2));
        if x1 == 0
            x1 = 1;
        end
        x = rem(i/2,1);

        y1 = uint32(floor(j/2));
        y2 = uint32(ceil(j/2));
        if y1 == 0
            y1 = 1;
        end
        y = rem(j/2,1);
        ctl = img(x1,y1,:); %// Top-left
        cbl = img(x2,y1,:); %// Bottom-left
        ctr = img(x1,y2,:); %// Top-right
        cbr = img(x2,y2,:); %// Bottom-right
        %the bilinear formula in which we use to find correct mapping
        output(i,j,:) = ((cbr*y)+(cbl*(1-y))*x)+((ctr*y)+(ctl*(1-y))*(1-x));
    end
end
output = uint8(output);
end
```

As well as Pixel Replication..:

```
function output = pixel_replication(img)
 [X,Y] = size(img);
 %make sure output is 2* the size of the input
output = zeros(2*X,2*Y);
for x = 1:X
    for y = 1:Y
        %first we find the top-left, and then find others in respect to it
        j = 2*(x-1) + 1;
        i = 2*(y-1) + 1;
        output(j,i) = img(x,y); %// Top-left
        output(j+1,i) = img(x,y); %// Bottom-left
        output(j,i+1) = img(x,y); %// Top-right
        output(j+1,i+1) = img(x,y); %// Bottom-right
    end
end
output = uint8(output);
end
```

And finally,the main code:

```
clear;
syms c1 c2 c3 c4
%THE REFRENCE IMAGE
in1 = imread('C:\Users\Arya\Desktop\computer vision\HW\Images\1\car1.jpg');

%THE INPUT IMAGE
in2 = imread('C:\Users\Arya\Desktop\computer vision\HW\Images\1\car2.jpg');

figure(1);
imshow(in1);

figure(2);
imshow(in2);

% ----STEP 1 - SOLVE LINEAR EQUATION FOR THE 8 POINTS -----
eqn1 = 279*c1 + 512*c2+ (279*512)*c3+c4 == 697;
eqn2 = 239*c1 + 490*c2+ (239*490)*c3+c4 == 657;
eqn3 = 108*c1 + 402*c2+ (108*402)*c3+c4 == 532;
eqn4 = 336*c1 + 356*c2+ (336*356)*c3+c4 ==752;
[A,B] = equationsToMatrix([eqn1, eqn2, eqn3,eqn4], [c1, c2, c3,c4]);
cgp1  = round(linsolve(A,B)) %CONTAINS C1 TO C4

eqn1 = 279*c1 + 512*c2 + (279*512)*c3 + c4 == 483;
eqn2 = 239*c1 + 490*c2 + (239*490)*c3 + c4 == 441;
eqn3 = 108*c1 + 402*c2 + (108*402)*c3 + c4 == 385;
eqn4 = 336*c1 + 356*c2 + (336*356)*c3 + c4 == 337;
[A,B] = equationsToMatrix([eqn1, eqn2, eqn3,eqn4], [c1, c2, c3,c4]);
cgp2 = round(linsolve(A,B)) %CONTAINS C5 TO C8

%---- STEP 2 TRANSFORM INPUTS BASED ON NEWLY FOUND WEIGHTS FOR THE EQS -----

[V,W] = size(in2);
output = zeros(2*V,2*W);
midX = V;
midY = W;
z=0
for v = 1:V
    for w = 1:W
        i = V-(cgp1(1,1)*v + cgp1(2,1)*w + cgp1(3,1)*(v*w) + cgp1(4,1));

        j = W-(cgp2(1,1)*v + cgp2(2,1)*w + cgp2(3,1)*(v*w) + cgp2(4,1));

        output(i,j) = in2(v,w);
    end
        z = z+1
end

output = uint8(output);
```

The code which I finally used, modifies both images manually and then stiches them together:

```
in1 = imread('C:\Users\Arya\Desktop\computer vision\HW\Images\1\car1.jpg');

in2 = imread('C:\Users\Arya\Desktop\computer vision\HW\Images\1\car2.jpg');

figure(1);
in1_modified = in1(1:end-17,1:777,:);
imshow(in1_modified);

in2_modified = in2(18:end,360:end,:);
figure(2);
% imshow(in2_modified);
imshow([in1_modified, in2_modified]);
```

Problem 1.1.3:

The code I used for rotation, with Comments to explain:

```
clear

im1 = imread('C:\Users\Arya\Desktop\computer vision\HW\Images\1\Elaine.bmp');

imshow(im1);
[m,n,p]=size(im1);


deg = 30;
%converting the degrees to radians..
thet = deg*pi/180;
%we take m*2 and n*2 to make sure the rotated image fits in the range
output_m = m*2;
output_n = n*2;
for t=1:output_m
    for s=1:output_n
        %Here we're doing the affine transformation as well as inverse
        %mapping, hence the reason for subtracting with output_m/2 = m and
        %output_n/2 = n
       i = round((t-m)*cos(thet)+(s-n)*sin(thet)+m/2);
       j = round((s-n)*cos(thet)+n/2-(t-m)*sin(thet));
       if i>0 && j>0 && i<=m && j<=n
          im2(t,s,:)=im1(i,j,:);
       end
    end
end
figure;
imshow(im2);
```

```
clear;

in = imread('C:\Users\Arya\Desktop\computer vision\HW\Images\1\Goldhill.bmp');
figure(1);
%below code downsamples by 2, by eliminating rows & cols with factor of 2.

ds_img_no_avg = in(1:2:end,1:2:end);

%below code takes an average between 2 rows & cols
ds_img_avg = in(ceil(mean(reshape(1:end,2,[]))), ceil(mean(reshape(1:end,2,[]))));

%and here below we represent, PI and BI
out1 = bilinear_interpolation(ds_img_no_avg);
subplot(1,2,1)
imshow(out1);
title('B-I using elim method')
BI_Immse = immse(in,out1)

out2 = pixel_replication(ds_img_no_avg);
subplot(1,2,2)
imshow(out2);
title('P-R using elim method')

PR_Immse = immse(in,out2)
```