Image Processing
lab 1

Klaas Kliffen Jan Kramer

November 26, 2015

Exercise 1 – Zooming and shrinking by pixel replication

a. There are two cases, one for shrinking and zooming. In case of shrinking we create a new image with the desired dimensions calculated from the factor. For each pixel in the new image, the corresponding pixel from the original image is sampled. In the case of zooming, the desired are calculated in a similar fashion. Instead of sampling, the corresponding pixel from the original is duplicated multiple times.

```
function out = IPresize(img,factor)

if (factor < 0)
  factor *= -1;
  new = uint8(zeros(size(img)/factor));

  for i = 1 : columns(new)
    for j = 1 : rows(new)
       new(j,i) = img(j*factor,i*factor);
    endfor
  endfor
else
  new = uint8(zeros(size(img)*factor));
  % because matlabs starts at 1....
  climit = columns(new)-factor;
  rlimit = rows(new)-factor;
  for i = 1 : climit
    for j = 1 : rlimit
      new(j,i) = img(floor(j/factor)+1,floor(i/factor)+1);
    endfor
  endfor
endif

out = new;
```

Figure 1: The resulting image of shrinking and then zooming with a factor 10

Figure 2: original image

b.

c. When you shrink the original image pixel information is lost, since only every tenth pixel is used for the new image. When zooming in, the information can not be retrieved, since the new image only contians the samples of every tenth pixel.

Exercise 2 – Histogram equalization

a. The histogram is created by iterating over all pixels and keeping of the number of occurences of each gray value. This 1D array is then plotted.

```
1
2  function hist = IPhistogram(img)
3
4  h = zeros(256,1);
5  height = rows(img);
6  width = columns(img);
7
8
9  for i = 1:height
10   for j = 1:width
```

```
    val = img(i,j) + 1;
    h(val)++;
  endfor
endfor

hist = h;
```

b. The function uses the histogram of the former exercise. It can be given as an argument or is created by the function itself. Then a cummalative histogram is created. The histogram is then normalised for the number of pixels. Then a new image is created by iterating over all the pixels in the original image. For each puixel, the value is looked up in the cummalative histrogram and then multiplied by 256 to acquire the new gray levels.

```
function outimg = IPhisteq(img,hist)

width = columns(img);
height = rows(img);

out = uint8(zeros(size(img)));
n = width * height;
rel = zeros(256,1);

% acquire historgram
if (nargin < 2)
  hist = IPhistogram(img);
endif

rel(1) = hist(1);
% make it cumalative
for i = 2 : 256
  rel(i) = rel(i-1) + hist(i);
endfor
% normalize it
rel = rel / n;
for i = 1 : height
  for j = 1 : width
    out(i,j) = uint8(round(256 * rel(img(i,j)+1)));
  endfor
endfor

outimg = out;
```

Figure 3: Original image

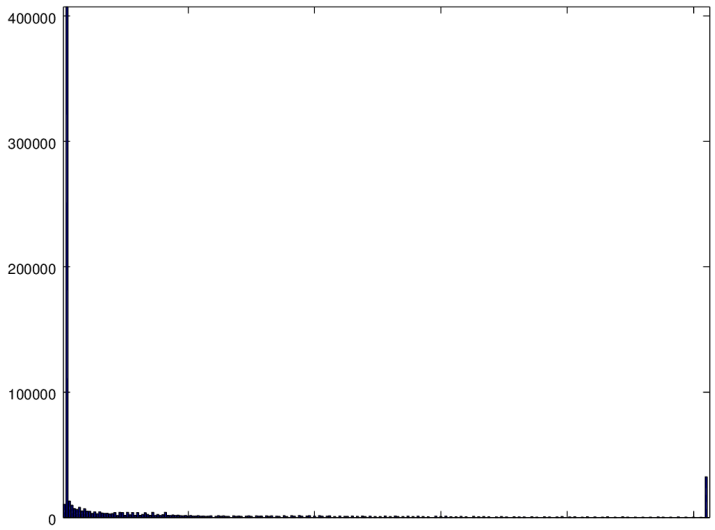

Figure 4: Histogram of the original image

Figure 5: Image after histogram equalization

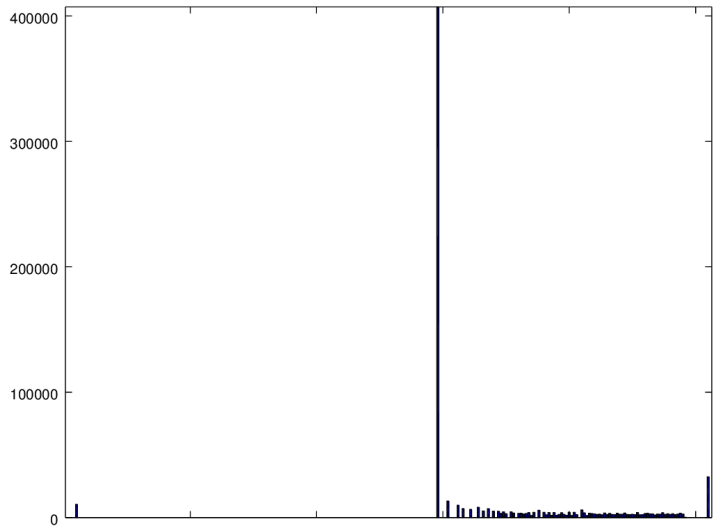

Figure 6: Histogram of the resulting image

The resulting image after histogram equalization has a wider spread of gray values, by making each gray level occur more equally. This allows for more contrast in the hard to see dark values on the leftside of the spine. The large peak in the centre of the histogram is the result of the large peak in the black values. This cannot be solved by the histogram equalization, because these values are all the same.

d. Applying the histogram equalization again would have no effect. Since the histogram after one pass of the algorithm already should make the histogram as flat as possible.

Exercise 3 – Spatial filtering

a. First, 8 images are created from the original image by shifting rows and columns in the 8 neighborhood. Then each part of the mask is applied seperately on each of these images in such that the each part of the mask is applied to the right (shifted) image. These values are then summed to achieve correlation of the image with the mask.

```
function [retval] = IPfilter (im, mask)

#first convert image data to double
i = im2double(im);

#get image dimensions
nr = size(i, 1);
nc = size(i, 2);

#now the shifted matrices can be computed like in the following grid:
# ul | u | ur
# l | i  | r
# dl | d | dr
#note that shifts are inversely related to the coordinates (x, y) of
    the pixel.
#for example a shift up means the the (x, y) position in the shifted
    matrix
#contains the (x, y + 1) pixel.
i_u = [i(2:nr, :); i(1, :)];
i_d = [i(nr, :); i(1:(nr - 1), :)];
i_l = [i(:, 2:nc) i(:, 1)];
i_r = [i(:, nc) i(:, 1:(nc - 1))];
i_ul = [i_u(:, 2:nc) i_u(:, 1)];
i_ur = [i_u(:, nc) i_u(:, 1:(nc - 1))];
i_dl = [i_d(:, 2:nc) i_d(:, 1)];
i_dr = [i_d(:, nc) i_d(:, 1:(nc - 1))];

#finally the mask can be applied (correlation)
pim = mask(1, 1) * i_dr + mask(1, 2) * i_d + mask(1, 3) * i_dl + ...
      mask(2, 1) * i_r  + mask(2, 2) * i   + mask(2, 3) * i_l  + ...
      mask(3, 1) * i_ur + mask(3, 2) * i_u + mask(3, 3) * i_ul;
#and the values can be converted back
retval = im2uint8(pim);

endfunction
```

b. This function uses the IPfilter function from the previous step. It applies the vertical and horizontal Sobel mask. The resulting image is retrieved by adding these togehter after taking the absolute value.

```
function [retval] = IPSobel (img)

#compute the horizontal and vertical differences
gx = IPfilter(img, [-1 -2 -1; 0 0 0; 1 2 1]);
gy = IPfilter(img, [-1 0 1; -2 0 2; -1 0 1]);

#did not check if output is bigger than uint8, should probably be
    done
retval = abs(gx) + abs(gy);

endfunction
```

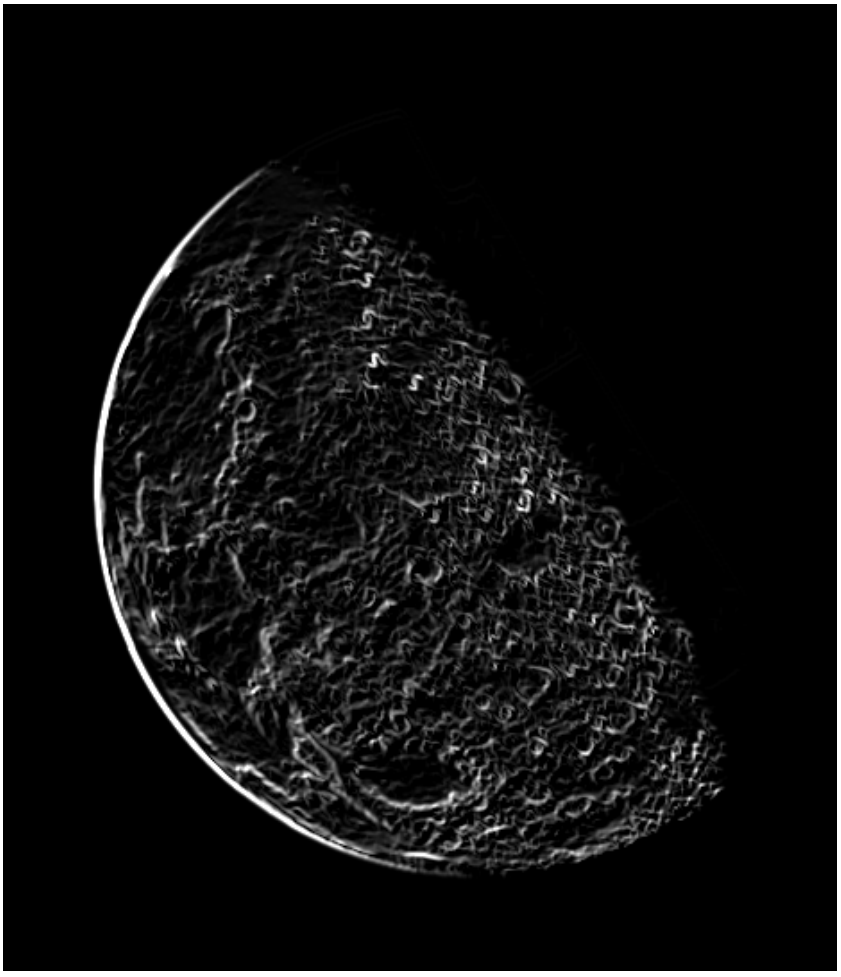

Figure 7: The result of the application of the IPSobel function to blurrymoon.tif

Task distribution

Name	ex1	ex2	ex3	report
Klaas	80%	80%	20%	50%
Jan	20%	20%	80%	50%