

Image Processing

lab 1

Klaas Kliffen Jan Kramer

November 27, 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.

```
1 function out = IPresize(img,factor)
2
3 if (factor < 0)
4     factor *= -1;
5     new = uint8(zeros(size(img)/factor));
6
7     for i = 1 : columns(new)
8         for j = 1 : rows(new)
9             new(j,i) = img(j*factor,i*factor);
10        endfor
11    endfor
12 else
13     new = uint8(zeros(size(img)*factor));
14     % because matlabs starts at 1....
15     climit = columns(new)-factor;
16     rlimit = rows(new)-factor;
17     for i = 1 : climit
18         for j = 1 : rlimit
19             new(j,i) = img(floor(j/factor)+1,floor(i/factor)+1);
20        endfor
21    endfor
22 endif
23
24 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 contains 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 occurrences 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

```

```

11     val = img(i,j) + 1;
12     h(val)++;
13     endfor
14 endfor
15
16 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 cumulative 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 pixel, the value is looked up in the cumulative histogram and then multiplied by 256 to acquire the new gray levels.

```

1  function outimg = IPhisteq(img,hist)
2
3  width = columns(img);
4  height = rows(img);
5
6  out = uint8(zeros(size(img)));
7  n = width * height;
8  rel = zeros(256,1);
9
10 % acquire histogram
11 if (nargin < 2)
12     hist = IPhistogram(img);
13 endif
14
15 rel(1) = hist(1);
16 % make it cumulative
17 for i = 2 : 256
18     rel(i) = rel(i-1) + hist(i);
19 endfor
20 % normalize it
21 rel = rel / n;
22 for i = 1 : height
23     for j = 1 : width
24         out(i,j) = uint8(round(256 * rel(img(i,j)+1)));
25     endfor
26 endfor
27
28 outimg = out;

```



Figure 3: Original image

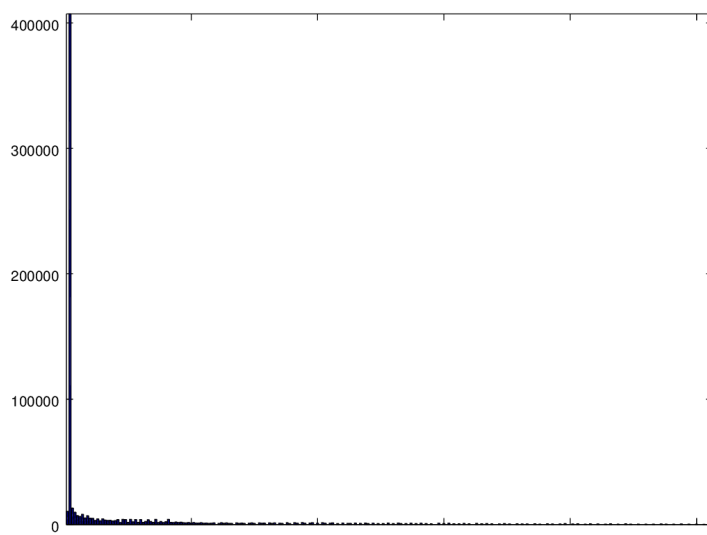


Figure 4: Histogram of the original image



Figure 5: Image after histogram equalization

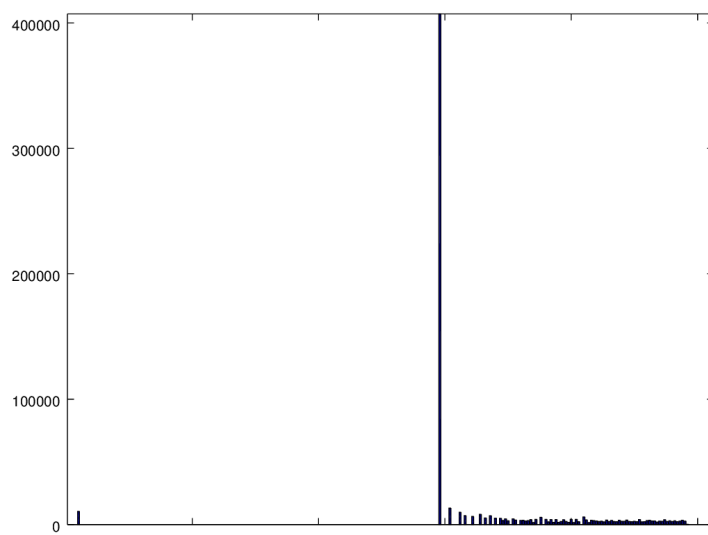


Figure 6: Histogram of the resulting image

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. The algorithm is based on the algorithm shown in the slides for applying a uniform mask. 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.

```

1  function [retval] = IPfilter (im, mask)
2
3  #first convert image data to double
4  i = im2double(im);
5
6  #get image dimensions
7  nr = size(i, 1);
8  nc = size(i, 2);
9
10 #now the shifted matrices can be computed like in the following grid:
11 # ul | u | ur
12 # l | i | r
13 # dl | d | dr
14 #note that shifts are inversely related to the coordinates (x, y) of
   the pixel.
15 #for example a shift up means the the (x, y) position in the shifted
   matrix
16 #contains the (x, y + 1) pixel.
17 i_u = [i(2:nr, :); i(1, :)];
18 i_d = [i(nr, :); i(1:(nr - 1), :)];
19 i_l = [i(:, 2:nc) i(:, 1)];
20 i_r = [i(:, nc) i(:, 1:(nc - 1))];
21 i_ul = [i_u(:, 2:nc) i_u(:, 1)];
22 i_ur = [i_u(:, nc) i_u(:, 1:(nc - 1))];
23 i_dl = [i_d(:, 2:nc) i_d(:, 1)];
24 i_dr = [i_d(:, nc) i_d(:, 1:(nc - 1))];
25
26 #finally the mask can be applied (correlation)
27 pim = mask(1, 1) * i_dr + mask(1, 2) * i_d + mask(1, 3) * i_dl + ...
28       mask(2, 1) * i_r + mask(2, 2) * i + mask(2, 3) * i_l + ...
29       mask(3, 1) * i_ur + mask(3, 2) * i_u + mask(3, 3) * i_ul;
30 #and the values can be converted back
31 retval = im2uint8(pim);
32
33 endfunction

```

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

```
1 function [retval] = IPSobel (img)
2
3 #compute the horizontal and vertical differences
4 gx = IPfilter(img, [-1 -2 -1; 0 0 0; 1 2 1]);
5 gy = IPfilter(img, [-1 0 1; -2 0 2; -1 0 1]);
6
7 #did not check if output is bigger than uint8, should probably be
   done
8 retval = abs(gx) + abs(gy);
9
10 endfunction
```

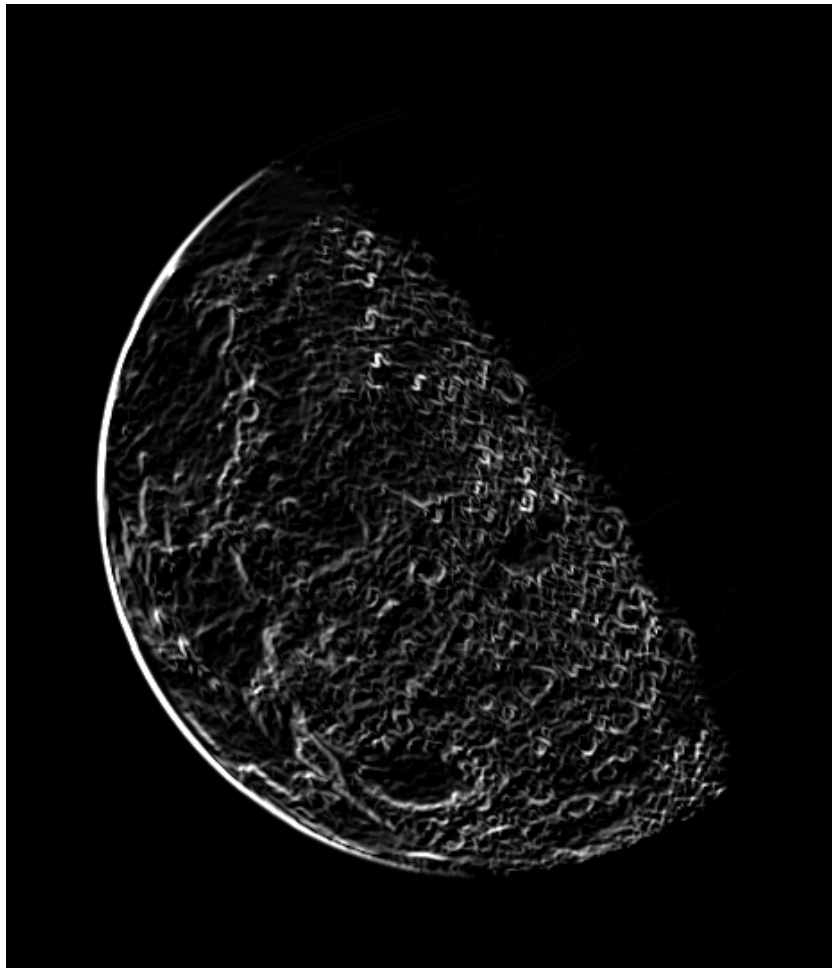


Figure 7: The result of the application of the IPSobel function to blurrymoon.tif

Task distribution

ex1	design	implementation	answers questions	writing report
Klaas	80%	80%	50%	50%
Jan	20%	20%	50%	50%

ex2	design	implementation	answers questions	writing report
Klaas	80%	80%	50%	50%
Jan	20%	20%	50%	50%

ex3	design	implementation	answers questions	writing report
Klaas	20%	20%	50%	50%
Jan	80%	80%	50%	50%