Assignment 2
Signal & Image Processing

Casper B. Hansen
University of Copenhagen
Department of Computer Science
fvx507@alumni.ku.dk

September 11, 2014

Abstract

In this assignment we explore the histogram-based processing techniques, and image filtering and enhancement.

Contents

1 Histogram-based Processing

1.1 CDF in relation to PDF

The cumulative distribution function is the integral of the probabililty density function. It calculates the cumulative probability of intensity values in the interval $[0; x]$, for a given x.

1.2 Constant images

Under the assumption that a constant image refers to an image of unchanging signal — that is, an image consisting of a repeating pixel intensity. This would mean that the probability density at $p_x(k) = 1$ for k, whilst $\forall i \neq k(p_x(i) = 0)$. This would make a histogram with exactly one spike at $x = k$, and a cumulative distribution curve that would be zero for all $x < k$, and 1 for all $x \geq k$.

1.3 Implementing cumulative histogram

As we can see from the rendition of the histogram and the cumulative histogram shown in figure **??** there are two regions of fast increase in the curve, which indicate an increase in light of the image. Similarly, the flat region indicate a steady intensity signal.

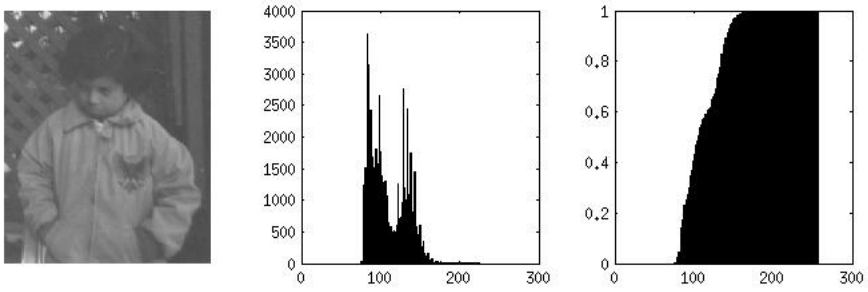

Figure 1: Original (left), histogram (middle) and cumulative histogram (right)

The code used to generate the figure is given in **??**.

1.4 Floating-point Image CDF

The code is given in **??**.

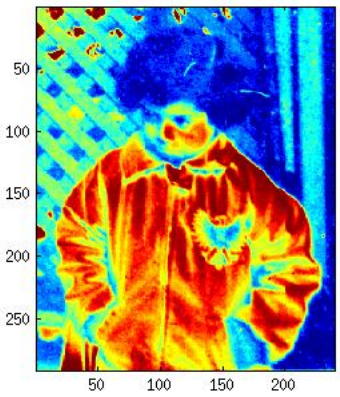

Figure 2: Rendition of the floating-point CDF

1.5 Invertibility of CDF

The CDF is not generally invertible. This is because of the possibly non-injective characteristics it may present. To overcome this, we may define a so-called *pseudo-inverse* within a given range — the code for calculating a pseudo-inverse for a given CDF is shown in **??**.

1.6 Histogram Matching

Following equation (3.25) [**?**, pp. 74] we arrive at an equivalent MatLab function shown in **??**.

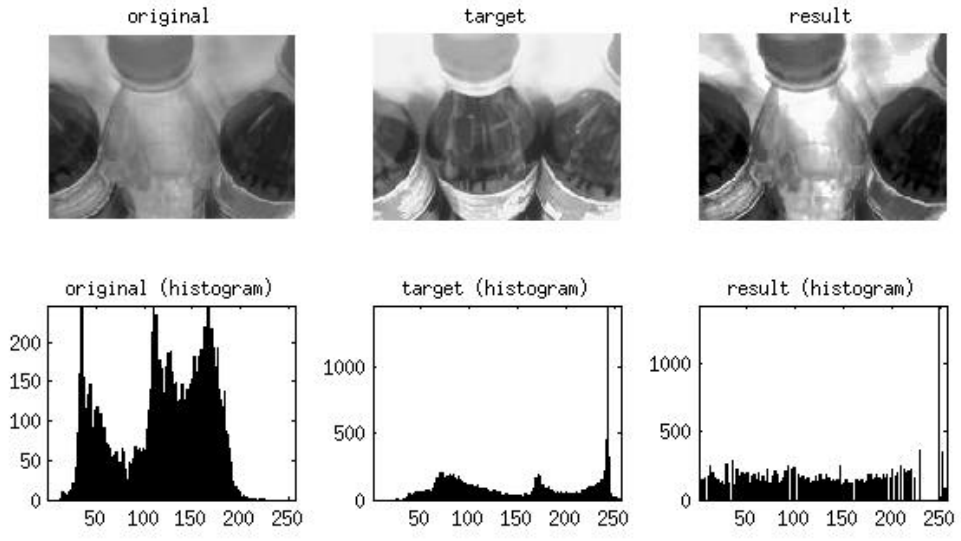

Figure 3: Histograms of the matching result

1.7 Matching Algorithm Comparison

Here we see the results of the previously discussed algorithm, the implementation of which can be reviewed in **??**, in comparison with the built-in MatLab function `histeq`.

Figure 4: Comparison of histogram matching algorithms

The program used to generate this figure can be reviewed in **??**

1.8 Prove the Midway

I didn't have time to do this one.

1.9 Simplistic Midway Algorithm

If we employ the more simplistic mathematical description of the algorithm, we can generalize it for a sequence of N images. Such an algorithm would simply calculate the average across all the cumulative histograms in the sequence.

$$C_m(x) = \frac{1}{N}\sum_{i=0}^{N} C_i(x) \tag{1}$$

An implementation of the midway specification can be reviewed in **??**.

2 Image Filtering and Enhancement

2.1 Finite Difference Approximations

$$f(x,y) = \sum_{i=I_{\min}}^{I_{\max}} \sum_{j=J_{\min}}^{J_{\max}} w(i,j)\frac{I(i+x+1, j+y) - I(i+x-1, j+y)}{2} \tag{2}$$

$$f(x,y) = \sum_{i=I_{\min}}^{I_{\max}} \sum_{j=J_{\min}}^{J_{\max}} w(i,j)\frac{I(i+x, j+y+1) - I(i+x, j+y-1)}{2} \tag{3}$$

2.2 Linearly Separable Filters

These types of filters can be carried out row-by-row, and column-by-column, since less pixels are affecting the intermediate results, I would think this is what allows linearly separable filters to be more robust toward noise.

This is a guess, and I didn't have time to put more thought into it than that.

2.3 Comparison of Mean and Median Filtering

As we can see from figure **??** the median filter is much better at reducing *salt and pepper* noise, whilst the mean filter (at least in my opinion does a somewhat better job at reducing gaussian noise.

The code used to generate the graph on the left, and the image figures below can be found in the appendix section **??**.

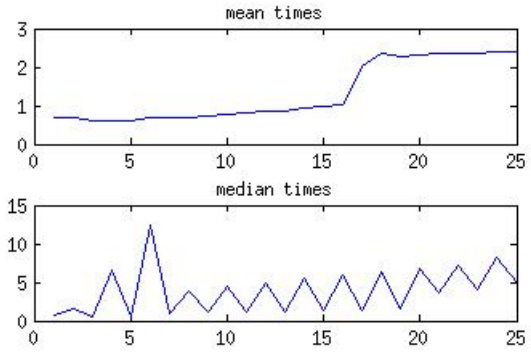

Figure 5: x-axis is N and y-axis is number of sec. for 100 computations

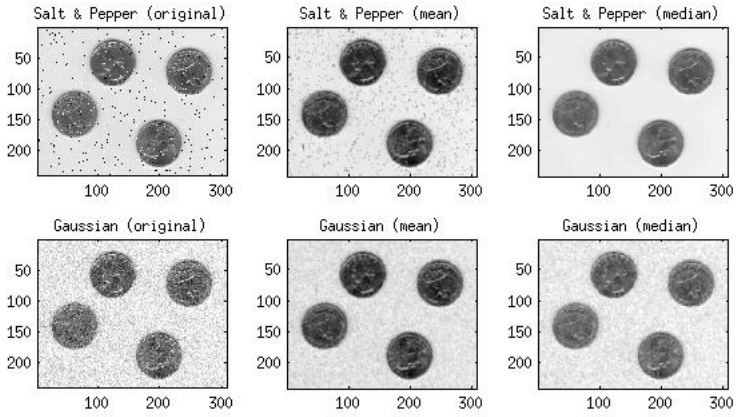

Figure 6: Showing originals (left), mean filtered (middle) and median filtered (right)

2.4 Filter Limits

As we increase N the region affected by the filter gets so large that the standard deviation σ no longer has any effect. In this case, I stop seeing any differences at approximately $N = 15$.

2.5 Standard Deviation

Keeping $N > 3\sigma$ we let the filter continue to let the imaginary pixels outside of the image affect the result, and as we continue to increase σ the image gradually moves towards a single uniform intensity across the entire image.

A Assignments

A.1 Cumulative Histogram

```
1  %% assignment 1.3
2
3  I = imread('images/pout.tif');
4  graph = histcum(I);
5
6  figure (1);
7  subplot(1,3,1), imshow(I), axis image;
8  subplot(1,3,2), bar(imhist(I)), axis on;
9  subplot(1,3,3), bar(graph), axis on;
```

Figure 7: Cumulative histogram (../assignment1.m)

A.2 Floating-point Image CDF

```
11 %% assignment 1.4
12
13 I = imread('images/pout.tif');
14 cdf = histcum(I);
15 FI = fpimg(I, cdf);
16 imagesc(FI), axis image;
17 colormap(jet);
```

Figure 8: Floating-point CDF (../assignment1.m)

A.3 Invertibility

```
19 %% assignment 1.5
20
21 I = imread('images/pout.tif');
22 cdf = histcum(I);
23 finv(cdf, 0.5);
```

Figure 9: CDF Invertibility (../assignment1.m)

A.4 Histogram Matching

```
25 %% assignment 1.6
26
27 I = rgb2grey(imread('images/cola1.png'));
28 T = rgb2grey(imread('images/cola2.png'));
29 R = histmatch(I, T);
30
31 subplot(2,3,1), imshow(I), axis image, title('original');
32 subplot(2,3,2), imshow(T), axis image, title('target');
33 subplot(2,3,3), imshow(R), axis image, title('result');
34
35 subplot(2,3,4), bar(imhist(I)), axis tight, title('original (histogram)');
36 subplot(2,3,5), bar(imhist(T)), axis tight, title('target (histogram)');
37 subplot(2,3,6), bar(imhist(R)), axis tight, title('result (histogram)');
```

Figure 10: Histogram matching (../assignment1.m)

A.5 Histogram Matching Comparison

Notice that we have to supply an image to our custom `histmatch` function — hence the difference in calculating the arguments. The results should however both follow the same histogram to match.

```
39 %% assignment 1.7
40
41 H = arrayfun( @(x) (1/256), 0:255);
42 CH = arrayfun( @(x) x * (1/255), 0:255);
43
44 I = imread('images/pout.tif');
45 R1 = histmatch(I, CH);
46 R2 = histeq(I, H);
47
48 subplot(1,3,1), imshow(I), axis image, title('original');
49 subplot(1,3,2), imshow(R1), axis image, title('histmatch');
50 subplot(1,3,3), imshow(R2), axis image, title('histeq');
```

Figure 11: Histogram matching comparison (../assignment1.m)

A.6 Simplistic Midway Algorithm

```
52 %% assignment 1.9
53
54 I1 = rgb2grey(imread('images/movie_flicker1.tif'));
55 I2 = rgb2grey(imread('images/movie_flicker2.tif'));
56 M = midway(I1, I2);
57
58 R1 = histeq(I1, M);
59 R2 = histeq(I2, M);
60
61 subplot(2,2,1), imagesc(I1), axis image, title('1st, original');
62 subplot(2,2,2), imagesc(I2), axis image, title('2nd, original');
63 subplot(2,2,3), imagesc(R1), axis image, title('1st, midway');
64 subplot(2,2,4), imagesc(R2), axis image, title('2nd, midway');
65 colormap(gray);
```

Figure 12: Simplistic midway algorithm (../assignment1.m)

A.7 Comparison of Filters

```
%% assignment 2.3

I = imread('images/eight.tif');

Isp = imnoise(I, 'salt & pepper', 0.03);
Igs = imnoise(I, 'gaussian', 0.02);

% mean filter (credit: matlab examples)
mean_times = 1:25
for N = 1:25
    tic;
    for i = 1:100
        k = ones(N,N) / (N*N);
        IMsp = imfilter(Isp, k);
        IMgs = imfilter(Igs, k);
    end
    mean_times(N) = toc;
end

% median filter (credit: matlab examples)
median_times = 1:25
for N = 1:25
    tic;
    for i = 1:100
        IMsp = medfilt2(Isp, [N N]);
        IMgs = medfilt2(Igs, [N N]);
    end
    median_times(N) = toc;
end

figure(1);
subplot(2,1,1), plot(1:25, mean_times), title('mean times');
subplot(2,1,2), plot(1:25, median_times), title('median times');

figure(2);
subplot(2,3,1), imagesc(Isp), axis image, title('Salt & Pepper (original)');
subplot(2,3,4), imagesc(Igs), axis image, title('Gaussian (original)');

N = 3
k = ones(N,N) / (N*N);
IMeanSp = imfilter(Isp, k);
IMeanGs = imfilter(Igs, k);
subplot(2,3,2), imagesc(IMeanSp), axis image, title('Salt & Pepper (mean)');
subplot(2,3,5), imagesc(IMeanGs), axis image, title('Gaussian (mean)');

IMedSp = medfilt2(Isp, [N N]);
IMedGs = medfilt2(Igs, [N N]);
subplot(2,3,3), imagesc(IMedSp), axis image, title('Salt & Pepper (median)');
subplot(2,3,6), imagesc(IMedGs), axis image, title('Gaussian (median)');
colormap(gray);
```

Figure 13: Filter comparison (../assignment2.m)

B Functions

B.1 RGB to Greyscale Conversion Function

Computing the grey-scale image of an RGB image is done by scaling each channel using estimated visual perception coefficients[**?**, pp. 11].

```
1 function [ out ] = rgb2grey( I )
2     out = I(:,:,1) * 0.2989 + I(:,:,2) * 0.5870 + I(:,:,3) * 0.1140;
3 end
```

Figure 14: RGB to Greyscale conversion (../rgb2grey.m)

B.2 Cumulative Histogram

To compute the cumulative histogram we use MatLab's `imhist` to calculate the image histogram, and the cumulative sum thereof, using MatLab's `cumsum`.

```
1 function [ out ] = histcum( I )
2     h = imhist(I);
3     out = cumsum(h/sum(h));
4 end
```

Figure 15: MatLab function that computes the cumulative histogram (../histcum.m)

B.3 Floating-point Image CDF

We extract the image data of I interpreting it as floating-point values, or `double`. Then we apply the given `cdf` function to each of these values, returning the result.

```
1 function [ out ] = fpimg( I, cdf )
2     ds = double(I);
3     out = arrayfun(@(x) cdf(x), ds);
4 end
```

Figure 16: MatLab function that computes the floating-point image (../fpimg.m)

B.4 Pseudo-inverse of a CDF

We declare a list of the discrete values $\{0, \ldots, 255\}$ on which f is defined. For each of these we then apply the `cdf` function, and find all the indices into this list for which the condition $f(s) \geq l$ holds. For each of these we then apply the previously calculated values. The minimum of this set is then returned.

```
1 function [ out ] = finv( cdf, l )
2     indices = find( arrayfun(@(x) x >= l, cdf) ); % indices where f(s) >= l
3     out = min(cdf(indices)); % minimum corresponds to the inverse
4 end
```

Figure 17: MatLab function that computes a pseudo-inverse of a given CDF (../finv.m)

B.5 History-matching

On lines 2–3 we calculate C_x and C_z, and following that C_z^{-1} based on these on line 4. This results in a CDF that is equivalent to that of equation (3.25) [**?**, pp. 74].

```
1 function [ out ] = histmatch( I, T )
2     Cz = histcum(T);
3     Cx = histcum(I);
4     CzInv = arrayfun(@(x) finv(Cz, x), Cx);
5     out = CzInv(I);
6 end
```

Figure 18: History matching algorithm (../histmatch.m)

B.6 Midway Specification

```
1 function [ out ] = midway( I1, I2 )
2     out = (histcum(I1) + histcum(I2)) ./ 2;
3 end
```

Figure 19: Midway specification algorithm (../midway.m)

References

[1] Solomon & Breckon, *Fundamentals of Digital Image Processing - A Practical Approach with Examples in Matlab*, 2011