# Assignment 2 - Signal and Image Processing 2014

Martin Jørgensen, tzk173

September 11, 2014

## Contents

# 1 Task 2.1 Histogram-based processing

### Task 2.1.1

The CDF is the integration of the PDF over certain value ranges. As such the PDF is the probability of a pixel becoming exactly "some value $x$", while the CDF is the probability that a pixel takes a value "smaller than or equal to $x$". Variations in the CDF will thus represent an increased probability for the distinct value occurring. This behaviour is similar to the histogram and cumulative histogram in Figure 1.

### Task 2.1.2

The PDF of a constant image would be 1 for the value that is constant in the image and 0 for all other. The CDF would be 0 until the constant value is reached, and then become 1 for the reminder of the range.

### Task 2.1.3

The code for this task can be found in the appendix. An increase in of the function means that the intensity value corresponding to the $X$ coordinate of the bar is present in the picture. The higher the increase the more frequent the value is in the image. Flat regions represent intensities that are not present in the image.

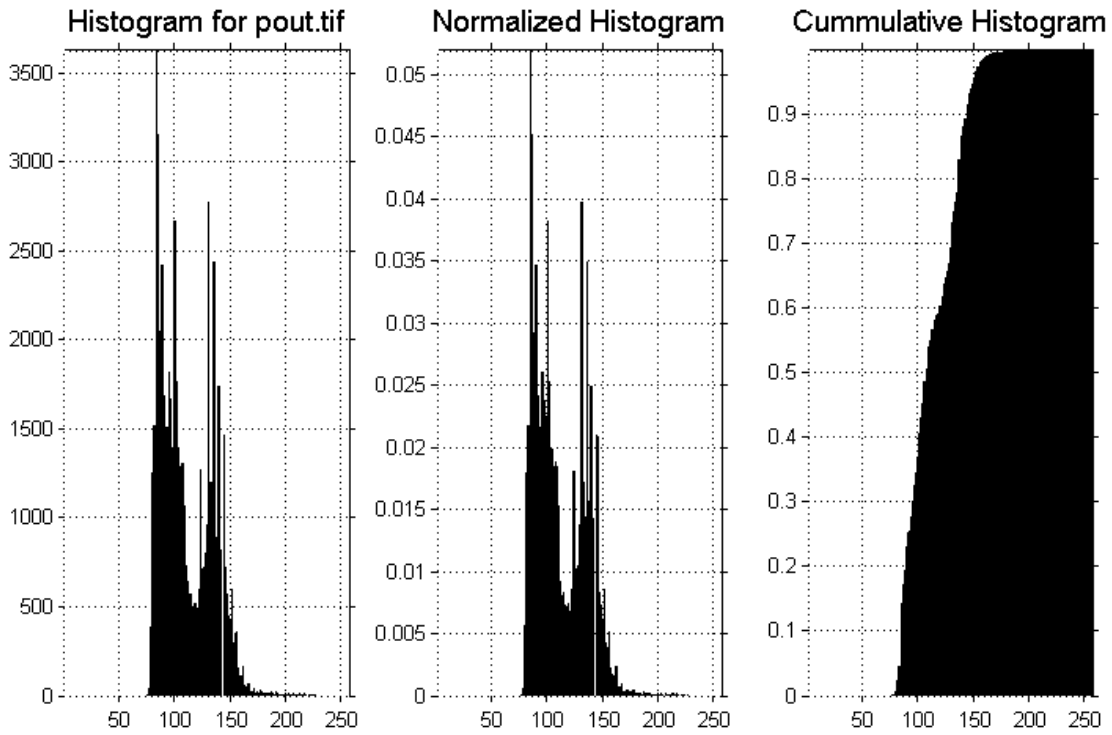



Figure 1: The histogram, normalized histogram and cumulative histogram for the file *pout.tif*.

### Task 2.1.4

Code can be found in the appendix. I created the function `fpi` that takes an image (or any Matlab array) and applies a CDF to it.

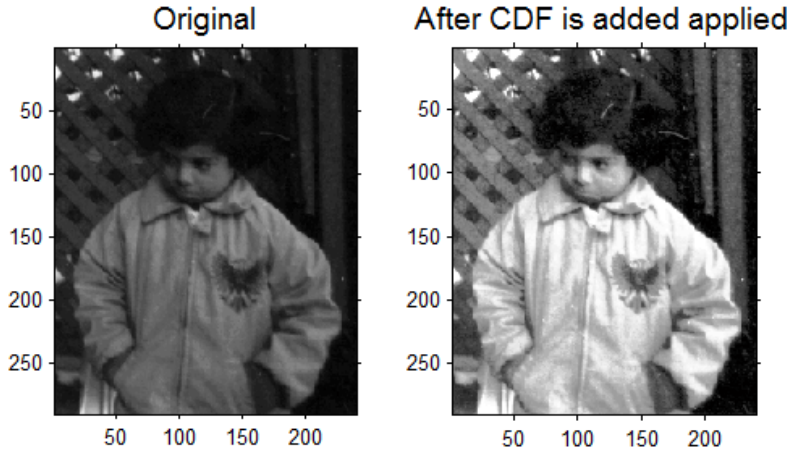



Figure 2: The picture *pout.tif*, before and after the CDF is added using `fpi`.

Figure 2 shows how applying an images CDF to the image itself using the `gpi` function will look for the image *pout.tif*.

### Task 2.1.5

Genrally the CDF is not invertable since several values can produce the same probability/result, it is not possible in all cases to calculate the value that caused a certain probability.

The code for the implementation can be found in the appendix. The solution is a function that given a function $f$ and a probability $l$ will find a value $s$ determined by $\min\{s|f(s) \geq l\}$. This is done by first acalculating $f(s)$ for all $s \in [0..255]$ then get the indexes for all the entries that are greater than or equal to $l$, pulling the $s$ values with that index and then picking the smallest using `min`.

The test was simply to pick $l = 0.0239$ and the CDF of *pout.tif* as CDF. The function returns 0.0240 which is the next entry in the CDF that is larger than our $l$.

### Task 2.1.6

Code is included in the appendix. For this assignment I created the function `histmatch` which tales two images as argument, a source image($C_x$ in the book.) and a target image ($C_z$ in the book.) and returns a copy of the source image, but with the matched histogram. The result of the testrun can be seen in Figure 3.

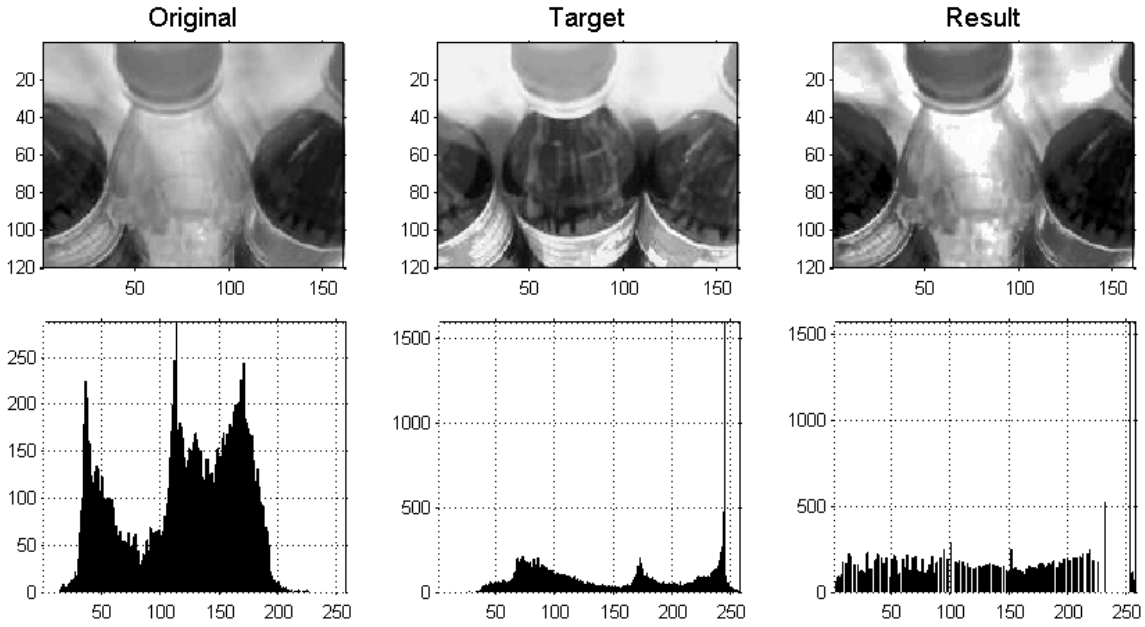



Figure 3: The original, target and combined images with their respective histograms.

### Task 2.1.7

The code for this task can be found in the appendix.

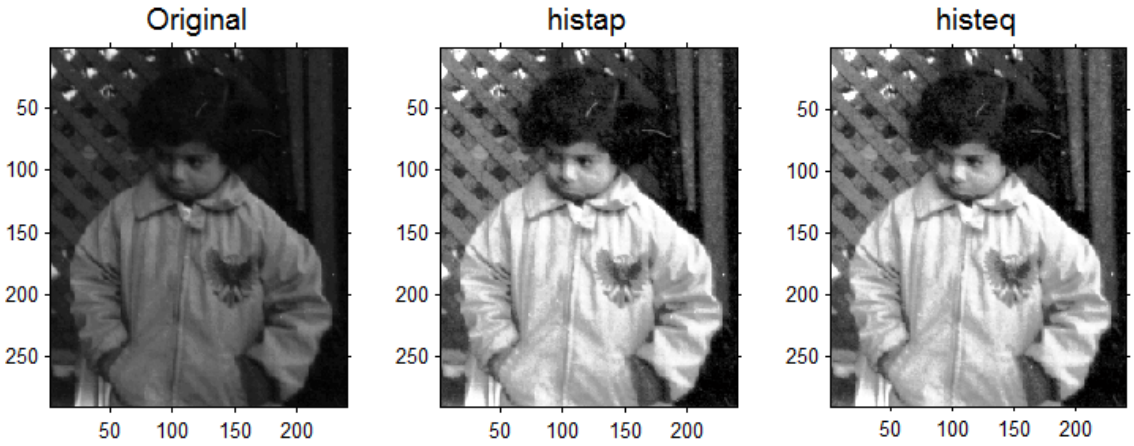



Figure 4: The original image and the results of applying the CDF using `histap` and `histeq`.

For this task I took some code from `histmatch` and created `histap` which applies a given histogram instead of calculating it first. As such the function is very similar.

### Task 2.1.8

I ran out of time before I could complete this task.

### Task 2.1.9

The code for this task can be found in the appendix.

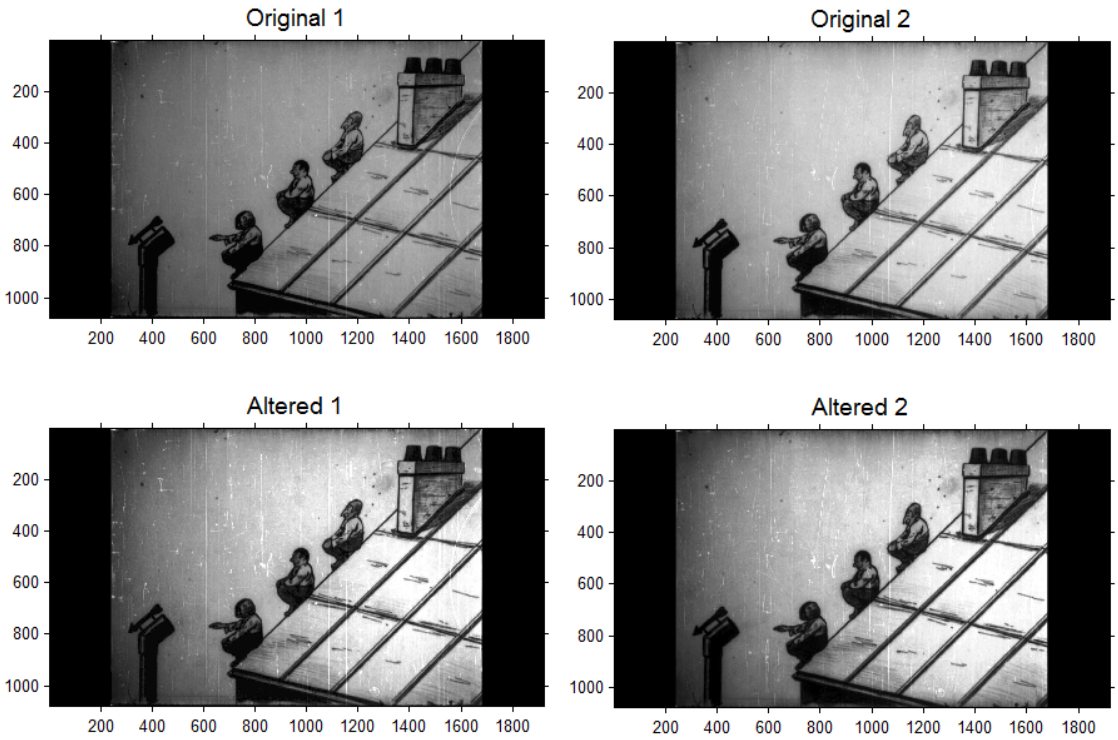



Figure 5: The two movies frames before and after the histogram alteration.

For this task I created a function called `midhist` which creates a cumulative histogram that corresponds to the average of the input histograms. It is then applied to the frames resulting in what you see in Figure 5.

For applying this technique to an arbitrary number of images the histogram could be computed like this

$$C_z = \frac{1}{N} \sum_{n=1}^{N} C_n(I_n)$$

Where $K$ is the total number of pictures and $I_n$ is the currently processing picture. $C_z$ should then be applied to all the pictures in the sequence.

## 2 Task 2.2 Image filtering and enhancement

### Task 2.2.1

Writing the partial derivatives on the form of eq 4.1 from the book will look like so:

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

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

I ran out of time before I could complete the rest of this task.

### Task 2.2.2

I ran out of time before I could complete this task.

### Task 2.2.3

The code for producing Figure 6 and the timings in Figure 7b and Figure 7a, can be found in the appendix.

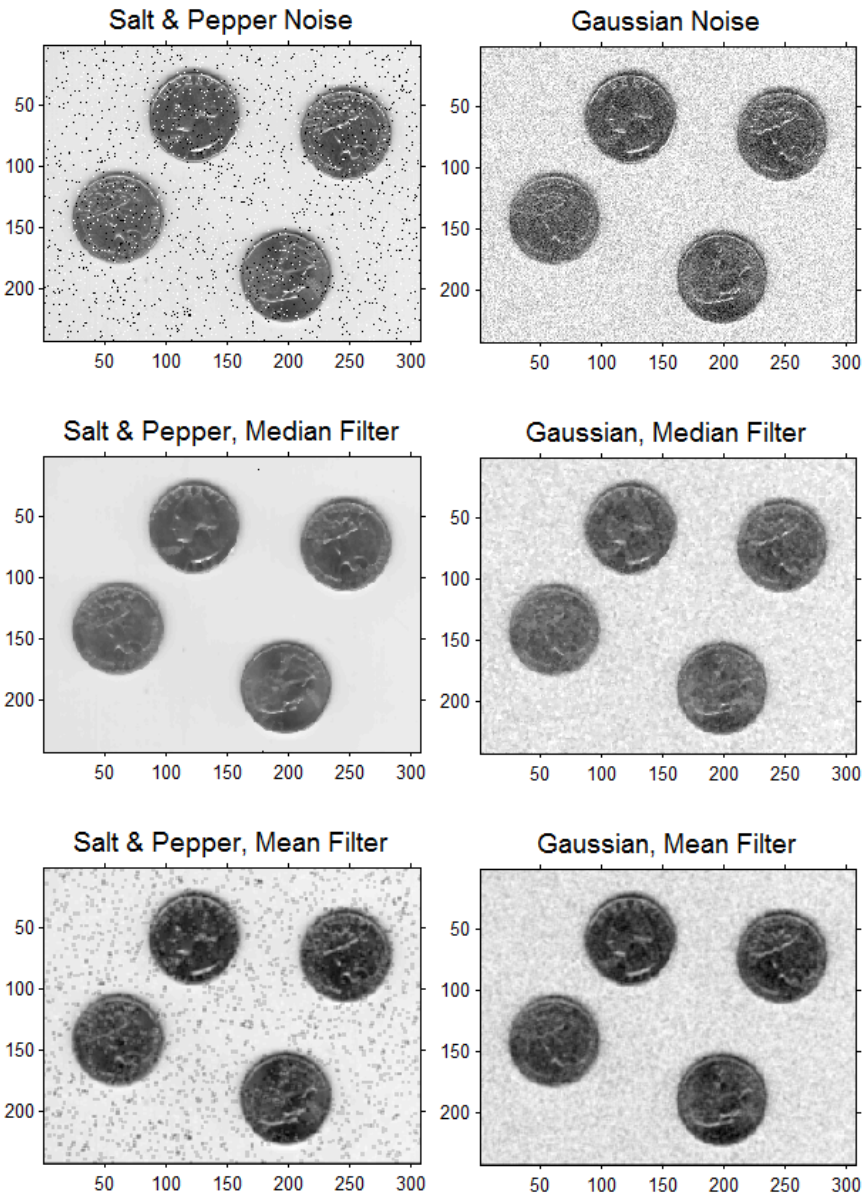
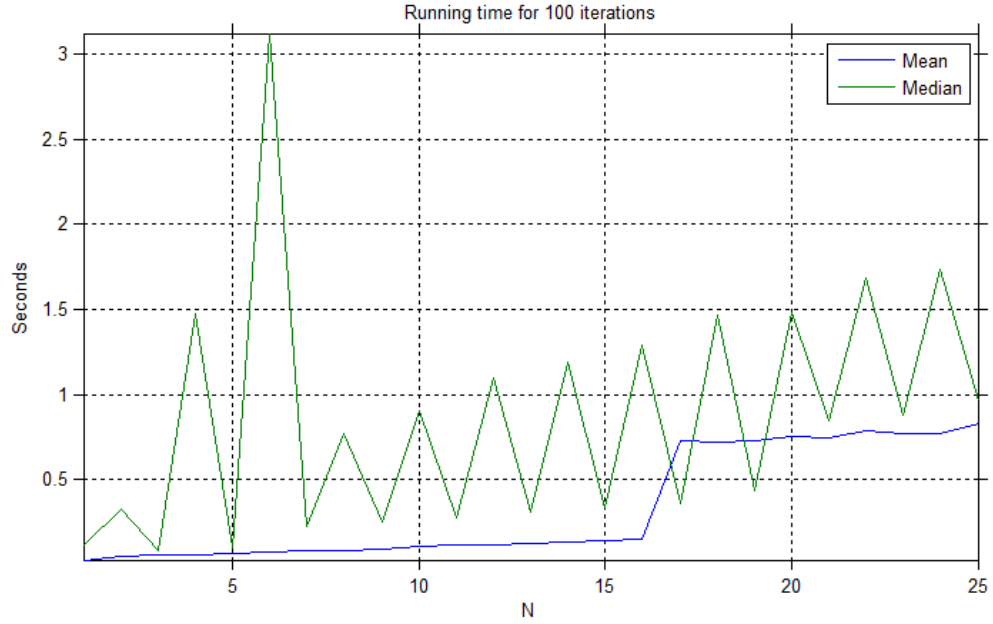



Figure 6: The image *eight.tif* with different types of noise and filters.

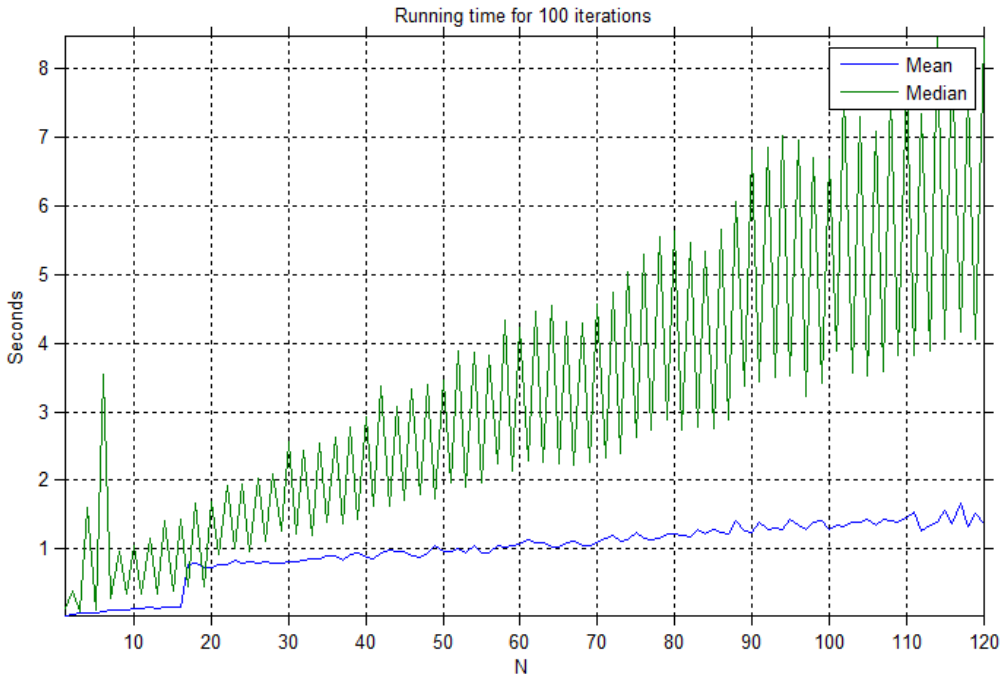
If the window size decides how big the kernel (and area of the image) is used to calculate the pixel values. Large windows sizes will thus tend to lower the contrast of an image since all pixels take "intensity cues" from the other pixels.

Figure 6 shows 2 different kinds of noise applied to the image *img/eight.tif*, and the result when applying different filters to the noisy images. Figure 7a and 7b shows plots of the time needed by each of the filters to run 100 repetitions for different values of $N$.

Figure 7a shows the running times for the specified number of $N$. But because I found the initial spikes in the Median running time suspicious I ran the test again, with $N$ values from




(a) For $N \in [1, 2, .., 25]$.




(b) For $N \in [1, 2, .., 120]$.

Figure 7

0 to 120, both to better see how the running time evolves, and also to see if the high spike is a pattern or singular. As seen on Figure 7b the spike is singular for the beginning, but the constant increase/decrease cycle is a pattern for the Median filter, albeit with an increase in amplitude and constant rise in running time. The Mean filter shows much less variance and have pretty low running time until $N \geq 17$ where it increases sharply and settles down again.

### Task 2.2.4

The code for producing Figure 8 is found in the appendix. When $N$ increases over a certain threshold, the image no longer changes. This happens because even though we increase the window size, we do not increase the $\sigma$-value, the weights around the center will decay to zero before reaching the end of the window. Increasing sigma along with window size will yield different results. As we shall see in the next section/task, increasing the $\sigma$-value along with the window size will cause the image to blur more and more.

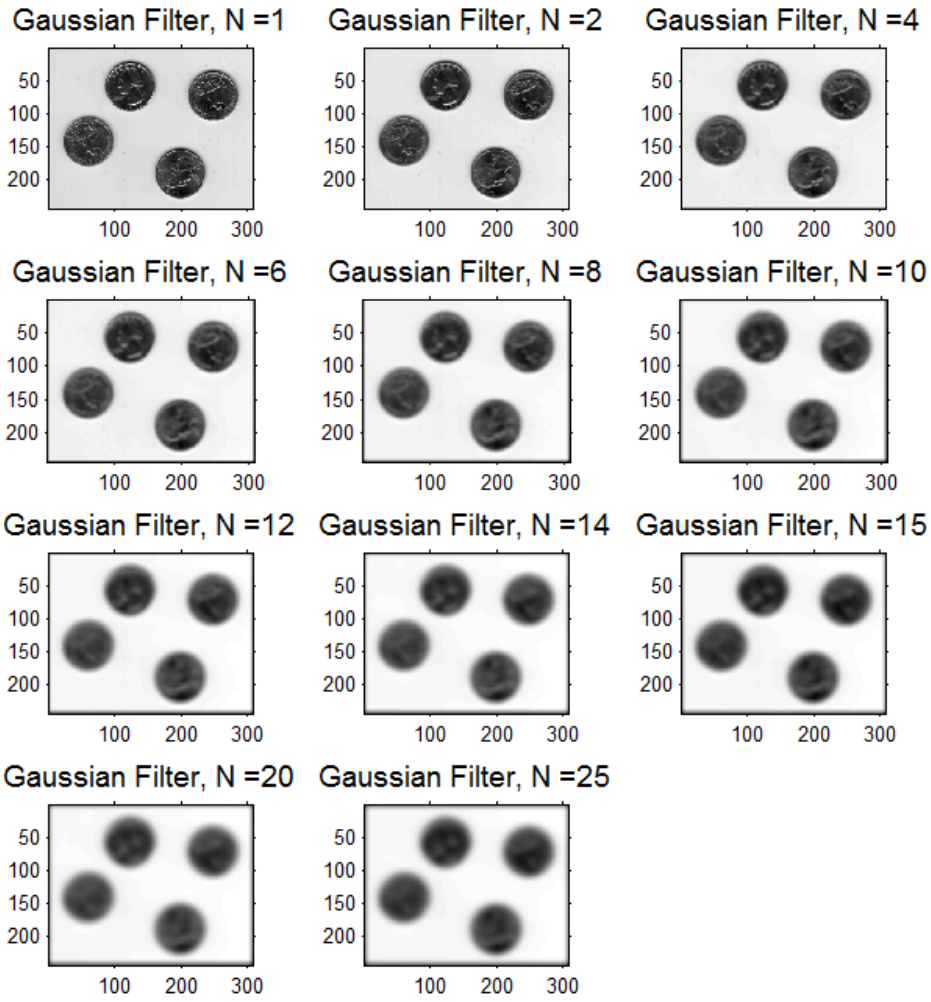



Figure 8: The image *eight.tif* with a Gaussian filter over it, using different values of $N$.

### Task 2.2.5

Code for producing Figure 9 can be found in the appendix.

Here I have tried different $\sigma$-values and increased the window size such that $N = 3 \times \sigma$. The result, as mentioned above is that the image blurs more and more until the shapes becomes undistinguishable.

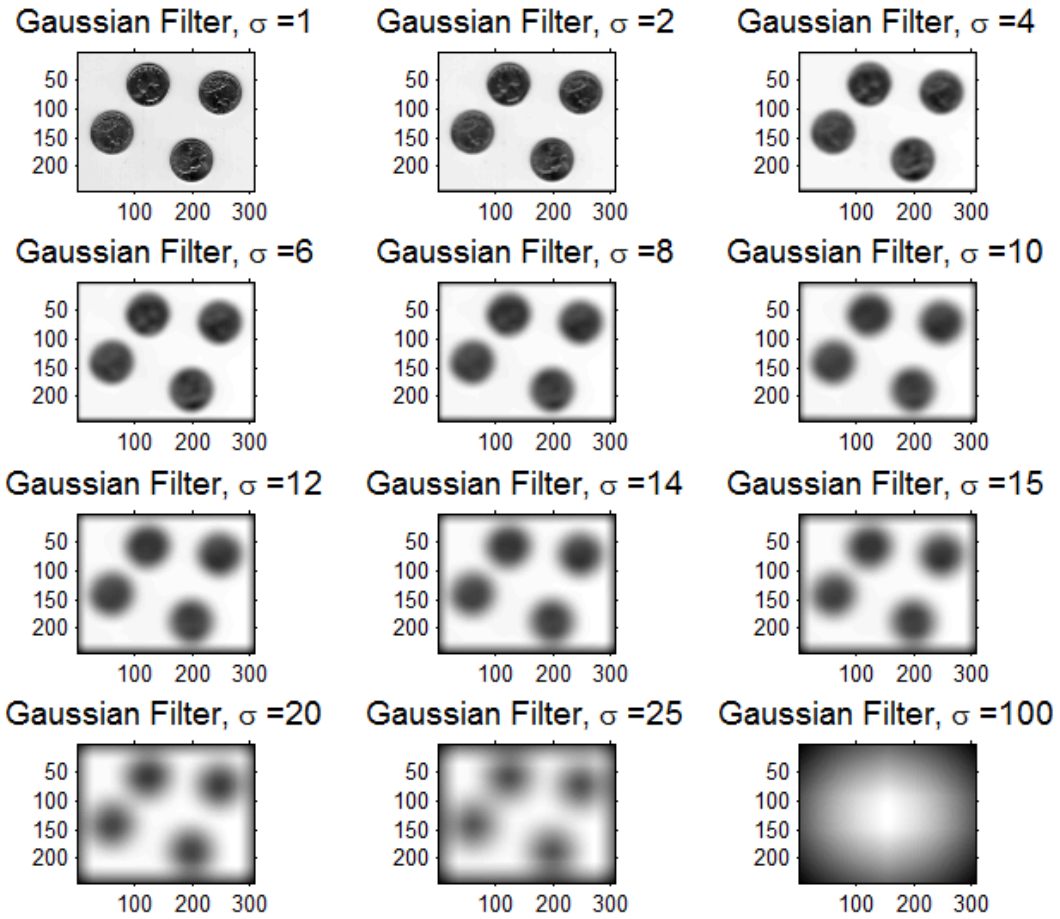



Figure 9: Different $\sigma$-values and their impact on the filtering.

# A Appendix

# B Task Code

### Task 2.1.3 Code

```
% Solution for part 1.3 of Assignment 2.
% Written by: Martin Jrgensen, tzk173

clear all;

I = imread('pout.tif');

% Create histogram and normalize.
hist = imhist(I);

% Create cumulative histogram.
cumhist = cumhist(hist);

% Also make the normalized
normhist = hist/sum(hist);

% Display side by side.
h = figure(213); set(h,'Color','White');
subplot(1,3,1); bar(hist,'FaceColor','black'); axis tight ; grid on;
set(gca,'TickDir','out');
title('Histogram for pout.tif','FontSize',14);

subplot(1,3,2); bar(normhist,'FaceColor','black'); axis tight ; grid on;
set(gca,'TickDir','out');
title('Normalized Histogram','FontSize',14);

subplot(1,3,3); bar(cumhist,'FaceColor','black'); axis tight ; grid on;
set(gca,'TickDir','out');
title('Cummulative Histogram','FontSize',14);
```

Figure 10: Code for solution. (../p13.m)

### Task 2.1.4 Code

```
% Solution for part 1.4 of Assignment 2.
% Written by: Martin Jrgensen, tzk173

clear all;

I1 = imread('pout.tif');

C = cumhist(imhist(I1));

I2 = fpi(I1, C);

h = figure(214); colormap(gray); set(h,'Color','White');
subplot(1,2,1); imagesc(I1); axis image; set(gca,'TickDir','out');
title('Original','FontSize',14);
subplot(1,2,2); imagesc(I2); axis image; set(gca,'TickDir','out');
title('After CDF is added applied','FontSize',14);
```

Figure 11: Code for solution. (../p14.m)

### Task 2.1.5 Code

```
% Solution for part 1.5 of Assignment 2.
% Written by: Martin Jrgensen, tzk173

clear all;

I = imread('pout.tif');
H = imhist(I);
CDF = cumhist(H);

finv(CDF,0.0239)
```

Figure 12: Code for solution. (../p15.m)

### Task 2.1.6 Code

```
% Solution for part 1.6 of Assignment 2.
% Written by: Martin Jrgensen, tzk173

clear all;

I1 = rgb2gray(imread('cola1.png'));
I2 = rgb2gray(imread('cola2.png'));
I3 = histmatch(I1, I2);

h = figure(216); colormap(gray); set(h,'Color','White');
subplot(2,3,1); imagesc(I1); axis image; set(gca,'TickDir','out');
title('Original','FontSize',14);

subplot(2,3,2); imagesc(I2); axis image; set(gca,'TickDir','out');
title('Target','FontSize',14);

subplot(2,3,3); imagesc(I3); axis image; set(gca,'TickDir','out');
title('Result','FontSize',14);

subplot(2,3,4); bar(imhist(I1),'FaceColor','black'); axis tight ; grid on; set(
    gca,'TickDir','out');
subplot(2,3,5); bar(imhist(I2),'FaceColor','black'); axis tight ; grid on; set(
    gca,'TickDir','out');
subplot(2,3,6); bar(imhist(I3),'FaceColor','black'); axis tight ; grid on; set(
    gca,'TickDir','out');
```

Figure 13: Code for solution. (../p16.m)

### Task 2.1.7 Code

```
% Solution for part 1.7 of Assignment 2.
% Written by: Martin Jrgensen, tzk173

clear all;

I1 = imread('pout.tif');
c2 = 0:(1/255):1;

I2 = histap(I1,c2);

I3 = histeq(I1,c2);

h = figure(217); colormap(gray); set(h,'Color','White');
subplot(1,3,1); imagesc(I1); axis image; set(gca,'TickDir','out');
title('Original','FontSize',14);

subplot(1,3,2); imagesc(I2); axis image; set(gca,'TickDir','out');
title('histap','FontSize',14);

subplot(1,3,3); imagesc(I2); axis image; set(gca,'TickDir','out');
title('histeq','FontSize',14);
```

Figure 14: Code for solution. (../p17.m)

### Task 2.1.9 Code

```
% Solution for part 1.9 of Assignment 2.
% Written by: Martin Jrgensen, tzk173

clear all;


I1 = rgb2gray(imread('movie_flicker1.tif'));
I2 = rgb2gray(imread('movie_flicker2.tif'));

C3 = midhist(I1, I2);
I3 = histeq(I1,C3);
I4 = histeq(I2,C3);


h = figure(219); colormap(gray); set(h,'Color','White');
subplot(2,2,1); imagesc(I1); axis image; set(gca,'TickDir','out');
title('Original 1','FontSize',14);
subplot(2,2,2); imagesc(I2); axis image; set(gca,'TickDir','out');
title('Original 2','FontSize',14);
subplot(2,2,3); imagesc(I3); axis image; set(gca,'TickDir','out');
title('Altered 1','FontSize',14);
subplot(2,2,4); imagesc(I4); axis image; set(gca,'TickDir','out');
title('Altered 2','FontSize',14);
```

Figure 15: Code for solution. (../p19.m)

### Task 2.2.3 Code

```
% Solution for part 2.4 of Assignment 2.
% Written by: Martin Jrgensen, tzk173

clear all;

% Read base image and create mean filter.
I = imread('eight.tif');
mf = fspecial('average');

% Generate noisy and filtered images.
J1 = imnoise(I, 'salt & pepper');
J2 = imnoise(I, 'gaussian');
J3 = medfilt2(J1);
J4 = medfilt2(J2);
J5 = filter2(mf, J1);
J6 = filter2(mf, J2);

% Setup the figure.
h = figure(2231); set(h,'Color','White'); colormap(gray);

% Display subplots for SP noise.
subplot(3,2,1); imagesc(J1); axis image; set(gca,'TickDir','out');
title('Salt & Pepper Noise','FontSize',14);
subplot(3,2,3); imagesc(J3); axis image; set(gca,'TickDir','out');
title('Salt & Pepper, Median Filter','FontSize',14);
subplot(3,2,5); imagesc(J5); axis image; set(gca,'TickDir','out');
title('Salt & Pepper, Mean Filter','FontSize',14);

% Display subplots for Gaussian noise.
subplot(3,2,2); imagesc(J2); axis image; set(gca,'TickDir','out');
title('Gaussian Noise','FontSize',14);
subplot(3,2,4); imagesc(J4); axis image; set(gca,'TickDir','out');
title('Gaussian, Median Filter','FontSize',14);
subplot(3,2,6); imagesc(J6); axis image; set(gca,'TickDir','out');
title('Gaussian, Mean Filter','FontSize',14);


% Lets time the functions!
Nrange = 25; % Change this if you wanna check for more N values.
medianTimes = zeros(1,Nrange);
meanTimes =  zeros(1,Nrange);
parfor N=1:Nrange
    tic % How long does 100 median iterations take.
    for i=1:100
        medfilt2(J1,[N N]);
    end
    medianTimes(N) = toc;

    mf = fspecial('average', N);
    tic % How long does 100 mean iterations take.
    for i=1:100
        filter2(mf, J1);
    end
```

### Task 2.2.4 Code

```
% Solution for part 2.4 of Assignment 2.
% Written by: Martin Jrgensen, tzk173

clear all;

% Read base image and create mean filter.
I = imread('eight.tif');

h = figure(224); set(h,'Color','White'); colormap(gray);
i = 1
for N=[1 2 4 6 8 10 12 14 15 20 25]
    gf = fspecial('gaussian', N, 5);
    I2 = filter2(gf, I);
    subplot(4,3,i); imagesc(I2); axis image; set(gca,'TickDir','out');
    title(strcat('Gaussian Filter, N = ',num2str(N)),'FontSize',14);
    i = i + 1;
end
```

Figure 17: Code for solution. (../p24.m)

### Task 2.2.5 Code

```
% Solution for part 2.5 of Assignment 2.
% Written by: Martin Jrgensen, tzk173

clear all;

% Read base image and create mean filter.
I = imread('eight.tif');

h = figure(225); set(h,'Color','White'); colormap(gray);
i = 1;
for sigma=[1 2 4 6 8 10 12 14 15 20 25 100]
    gf = fspecial('gaussian', 3*sigma, sigma);
    I2 = filter2(gf, I);
    subplot(4,3,i); imagesc(I2); axis image; set(gca,'TickDir','out');
    title(strcat('Gaussian Filter, \sigma = ',num2str(sigma)),'FontSize',14);
    i = i + 1;
end
```

Figure 18: Code for solution. (../p25.m)

# C Functions

### cumhist code

```
function [ out ] = cumhist( hist )
%cumhist Creates the normalized cumulative histogram for a given grayscale
    histogram.

% Normalize
hist = hist/sum(hist);

% Create cumulative histogram.
out = cumsum(hist);

end
```

Figure 19: Code for `cumhist` function. (../cumhist.m)

### finv code

```
function [ s ] = finv( CDF , l )
%finv Finds the pseudoinverse of a CDF.

s = min(arrayfun(@(x) CDF(x), find(arrayfun(@(x) x >= l, CDF))));

end
```

Figure 20: Code for `finv` function. (../finv.m)

### fpi code

```
function [ CI ] = fpi( I, CDF )
%fpi Applies the CDF function to all pixels in the image.

CI = arrayfun(@(x) CDF(x), double(I));

end
```

Figure 21: Code for `fpi` function. (../fpi.m)

### histap code

```
function [ I2 ] = histap( I1, c2 )
%histap Apply the histogram H to the image I.
h1 = imhist(I1);
c1 = cumhist(h1);
c2inv = zeros(256,1);
for i=1:256
    l = c1(i);
    c2inv(i) = finv(c2,l);
end

I2 = c2inv(I1);

end
```

Figure 22: Code for `histap` function. (../histap.m)

### histmatch code

```
function [ out ] = histmatch ( I1, I2 )
%histmatch Perform historgam matching between H1 and H2.
    h1 = imhist(I1);
    c1 = cumhist(h1);

    h2 = imhist(I2);
    c2 = cumhist(h2);

    c2inv = zeros(256,1);
    for i=1:256
        l = c1(i);
        c2inv(i) = finv(c2,l);
    end

    out = c2inv(I1);
end
```

Figure 23: Code for `histmatch` function. (../histmatch.m)

### midhist code

```
function [ C3 ] = midhist( I1, I2 )
%midhist Produces the midway histogram of two images.

H1 = imhist(I1);
H2 = imhist(I2);
C1 = cumhist(H1);
C2 = cumhist(H2);
C3 = (C1 + C2) / 2;

end
```

Figure 24: Code for `midhist` function. (../midhist.m)