Table of Contents

Exercise 4.1

```
%a
% The reverb delay in seconds will be equal to the number of samples
%times the period of a sample, s*(1/fs)
%The impulse response is  h(n) = #[n] + A#[n-s]
%apply to h(n) = #[n] + 0.8#[n-8000]
h = [1 zeros(1,7999) 0.8];

%b

[tyb_orig,fs] = audioread('TreatYouBetter.wav');
tyb_reverb = reverb_conv(tyb_orig, 8000, 0.8);
%soundsc(tyb_reverb,fs);
audiowrite('tyb_reverb.wav',tyb_reverb,fs);

%d

tyb_tremolo = tremolo(tyb_orig, 10/fs, 0.3);
%soundsc(tyb_tremolo,fs);
audiowrite('tyb_tremolo.wav',tyb_tremolo,fs);
% the tremolo makes the audio amplitude follow a cyclic pattern, as
 defined
% by the cos portion of the tremolo formula. This leads to the
 percieved
% volume of the audio going up and down.

%e
tyb_faded = tremolo(tyb_orig, 1/length(tyb_orig), 1);
%soundsc(tyb_faded,fs)

%The audio fades in and out becuase the the amplitude of tremelo
 effect is
%1. This means that as the cos goes into the negative portion, it
 reduces
%the signal progressivley towards zero

%f

tyb_faded_N = [zeros(1,123480) tyb_faded];
soundsc(tyb_faded_N,fs);
audiowrite('tyb_faded_N.wav',tyb_faded_N,fs);
tyb_orig_N = [zeros(1,123480) tyb_orig.'];
tyb_N_faded = tremolo(tyb_orig_N, 1/length(tyb_orig), 1);
soundsc(tyb_N_faded,fs);
```

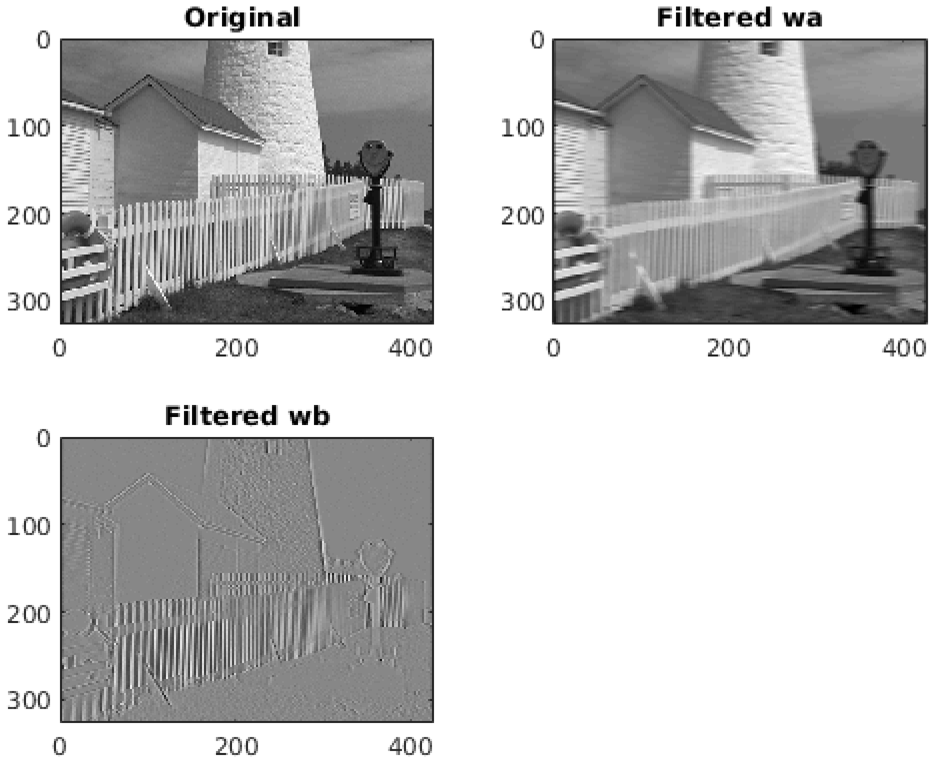
```
audiowrite('tyb_N_faded.wav',tyb_N_faded,fs);

% The two do not sound the same. One fades in at the beggining while
 one
% fades out. This is because the cos function changes the already
 faded
% audio in a different manner than it changes the original audio.
```

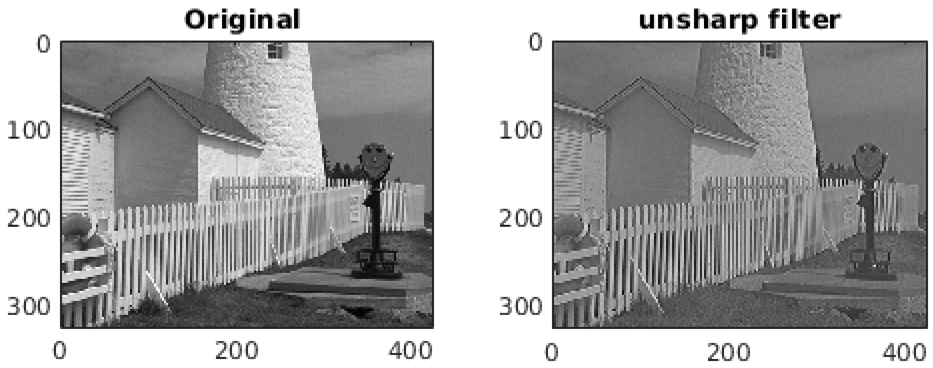
```
Warning: Data clipped when writing file.
Warning: Data clipped when writing file.
Warning: Data clipped when writing file.
```

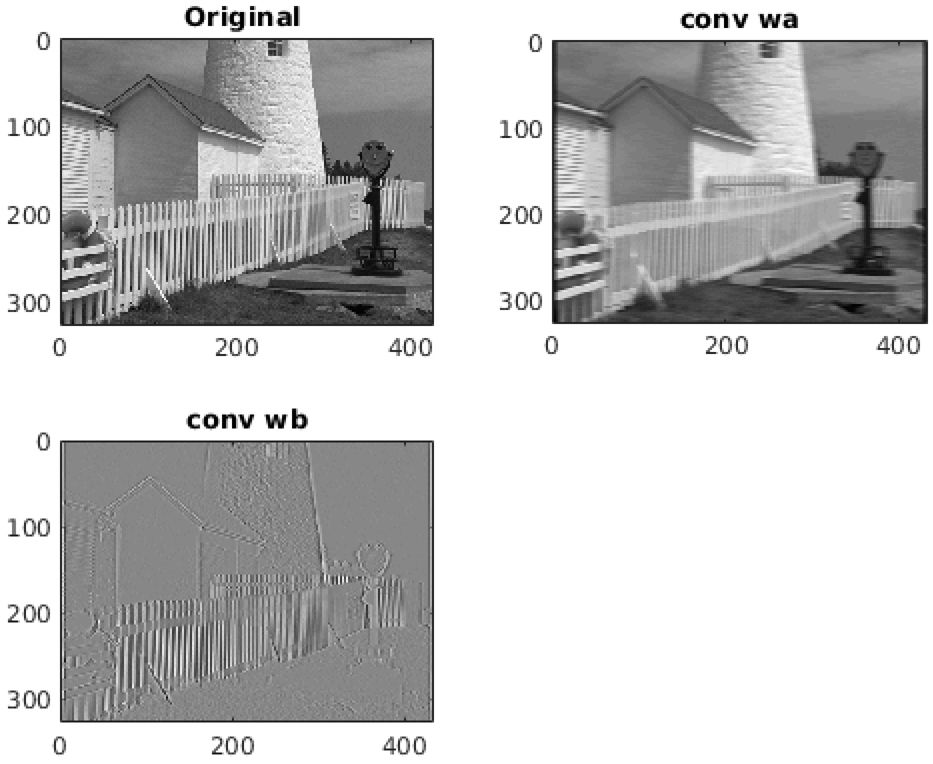
Exercise 4.2

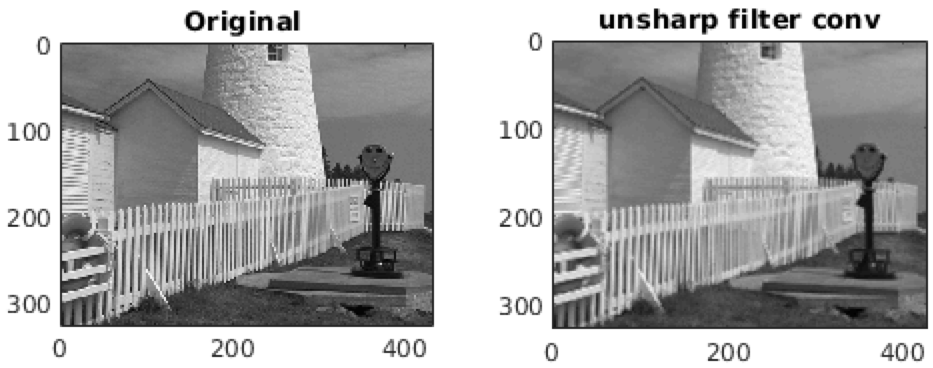
```
%a

wa = [1/9 1/9 1/9, 1/9 1/9 1/9, 1/9 1/9 1/9];
lighthouse = load('lighthouse.mat');
lighthouse_orig = lighthouse.lighthouse;
x = 0:1:size(lighthouse_orig,2);
y = 0:1:size(lighthouse_orig,1);

colormap('gray');

figure(1) ;
subplot (2 ,2 ,1) ;
imagesc (x , y , lighthouse_orig) ;
axis image ;
title ( 'Original ')

lighthouse_wa = filter2(wa,lighthouse_orig);

subplot (2 ,2 ,2) ;
imagesc (x , y , lighthouse_wa) ;
axis image ;
title ( 'Filtered wa')

% The filter blurs the image. This is known from the fact that the
 impulse
% response returned 1/9th for all values. Meaning that the original
 ones
% matrix was scaled by 1/9th. Coincidingly, the lighthouse image was
 scaled
% by 1/9th

%b

wb = [1/9 1/9 1/9, 1/9 -8/9 1/9, 1/9 1/9 1/9];

colormap('gray');

lighthouse_wb = filter2(wb,lighthouse_orig);
```

```
subplot (2 ,2 ,3) ;
imagesc (x , y , lighthouse_wb) ;
axis image ;
title ( 'Filtered wb')

% This filter extracted the edges from the image. This can be seen
 through
% the filter as the middle value in the kernal remains close to it's
% original absolute value, but the edges are significantly reduced.

%c

% the impulse response can be given by [8/9 8/9 8/9; 8/9 17/9 8/9; 8/9
 8/9
% 8/9

figure(2) ;
colormap('gray')
subplot (2 ,2 ,1) ;
imagesc (x , y , lighthouse_orig) ;
axis image ;
title ( 'Original ')

lighthouse_unsharp = image_unsharp_masking(lighthouse_orig);

subplot (2 ,2 ,2) ;
imagesc (x , y , lighthouse_unsharp) ;
axis image ;
title ( 'unsharp filter')

% This filter sharpens the image by making the edges of the image more
% pronounced. It is different from the other two images as one
 extracted
% edges and one blurred the image

%d

%da

figure(3) ;
colormap('gray');
subplot (2 ,2 ,1) ;
imagesc (x , y , lighthouse_orig) ;
axis image ;
title ( 'Original ')

length(x); % 427
length(y); % 327

lighthouse_wa = conv2(wa,lighthouse_orig);
x = 0:1:size(lighthouse_wa,2);
y = 0:1:size(lighthouse_wa,1);

length(x); %435
length(y); % 327

subplot (2 ,2 ,2) ;
imagesc (x , y , lighthouse_wa) ;
axis image ;
title ( 'conv wa')

% The size of the x axis was enlarged

%db

wb = [1/9 1/9 1/9, 1/9 -8/9 1/9, 1/9 1/9 1/9];

colormap('gray');

lighthouse_wb = conv2(wb,lighthouse_orig);
x = 0:1:size(lighthouse_wb,2);
y = 0:1:size(lighthouse_wb,1);

length(x); %435
length(y); % 327

subplot (2 ,2 ,3) ;
imagesc (x , y , lighthouse_wb) ;
axis image ;
title ( 'conv wb')

% The size of the convoluded image is larger in the x axis, in
 comparison
% to the original

%dc
figure(4) ;
colormap('gray')
subplot (2 ,2 ,1) ;
imagesc (x , y , lighthouse_orig) ;
axis image ;
title ( 'Original ')

lighthouse_unsharp = image_unsharp_masking_conv(lighthouse_orig);
x = 0:1:size(lighthouse_unsharp,2);
y = 0:1:size(lighthouse_unsharp,1);

length(x); %435
length(y); % 327

subplot (2 ,2 ,2) ;
imagesc (x , y , lighthouse_unsharp) ;
axis image ;
title ( 'unsharp filter conv')
```


Original
0
100
200
300
0
200
400
Filtered wa
0
100
200
300
0
200
400
Filtered wb
0
100
200
300
0
200
400


Original
unsharp filter
0
100
200
300
0
200
400


Original
conv wa
conv wb
0
100
200
300
0
200
400



FUNCTIONS USED

```
type reverb_conv
type reverb_filter
type reverb_own
type tremolo
type image_unsharp_masking
type image_unsharp_masking_conv
```

```
function yc = reverb_conv(x,s,A)
% This function uses the convolution method, which is computationally
% the same as multiplying polynomials whose coefficient values match
 those
% of h and x. It is more efficient then the filter, with a time of
%0.006 to execute
h = [1 zeros(1,s-1) A];
yc = conv(h,x);
end
```

```
function yc = reverb_filter(x,s,A)
% The filter function computationally uses the rational transfer
 function
% which can also be expressed as the function processing the
 difference
```

```
%eqn. It is not very efficient, with a time of 0.426 to execute
h = [1 zeros(1,s-1) A];
yc = filter(h,1,x);
end



function yo = reverb_own(x,s,A)
% This function uses a direct implementation, which is
 computationally
% equivalent to multiplying and adding the two functions. It is the
 most
%efficent with a time of 0.002 seconds to execute
x_delayed = [zeros(1,s) x.'];
x_longer = [x.' zeros(1,s)];
yo = x_longer + A*x_delayed;
end

function y = tremolo(x,fm,A)
L = length(x);
y = zeros(1,L);

for n = 1:L
    X = x(n) + A*cos(2*pi*fm*n)*x(n);
    y(n) = X;
end
end

function im_out = image_unsharp_masking(im_in)
wb = [1/9 1/9 1/9, 1/9 -8/9 1/9, 1/9 1/9 1/9];
im_temp = filter2(wb,im_in);
im_out = im_in - im_temp;
end


function im_out = image_unsharp_masking_conv(im_in)
%wb = [1/9 1/9 1/9, 1/9 -8/9 1/9, 1/9 1/9 1/9];
%im_temp = conv2(wb,im_in);
%spatial = [0 0 0; 0 1 0; 0 0 0];
spatial = [8/9 8/9 8/9; 8/9 -17/9 8/9; 8/9 8/9 8/9];
im_out = conv2(spatial,im_in);
%im_out = im_in - im_temp;
end
```

Published with MATLAB® R2020a