Koby Miller Lab4

Table of Contents

Exercise 4.1

```
%(a)
% Generating h for A = 0.8 and s = 8000
%A = 0.8;
%s = 8000
%
%a = [1];
%b = [1,A];
%
%x_size = size(x,1);
%impulse = zeros(1,x_size);
%impulse(1) = 1;
%
%h = filter(b,a,impulse);


% The 3 functions made here are printed at the bottom
% reverb_conv definitely has a higher computational efficiency as
% the output matrix is basically double of the outputs of the others



%(b)
[tyb_orig, fs] = audioread('TreatYouBetter.wav');

tyb_reverb = reverb_conv(tyb_orig, 8000, 0.8);
%tyb_reverb2 = reverb_filter(tyb_orig, 8000, 0.8);

%soundsc(tyb_orig, sample_rate)
%soundsc(tyb_reverb, sample_rate)
%soundsc(tyb_reverb2, sample_rate)

audiowrite('tyb_reverb.wav', tyb_reverb, fs);


%(c)
% funciton is shown at the bottom
```

```
%(d)
tyb_tremolo = tremolo(tyb_orig, 10/fs, 0.3);
%soundsc(tyb_tremolo,fs);
% The audio now sounds a little quieter. Like someone turned the
% volume down a bit. I can't really think why, if anything, I would
% think that it would raise the volume because you are adding the
% vector with a smaller portion of itself.
audiowrite('tyb_tremolo.wav', tyb_tremolo, fs);


%(e)
tyb_faded = tremolo(tyb_orig, 1/length(tyb_orig), 1);
%soundsc(tyb_faded,fs);
% now it sounds like the volume is turned down just a bit more. Again,
% I don't exactly know why it was faded in the first place but we
% did increase our tremolo amplitude, which seems to be related to
% how faded the output is.


%(f)
N_delay = zeros(1,123480);
tyb_faded_N = [N_delay';tyb_faded];
%soundsc(tyb_faded_N,fs);
audiowrite('tyb_faded_N.wav', tyb_faded_N, fs);

tyb_orig_N = [N_delay';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);

% They do sound the same which makes sense to me. For tyb_faded_N
% you are adding the tremolo effect and then delaying, whereas for
% tybe_N_faded you are delaying and then adding the tremolo effect.
% Delaying wouldn't cause tremolo to act any different. Maybe if our
% difference equation didn't have the 'n' inside the cos() than the
% delay would cause a drastic change.
```

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

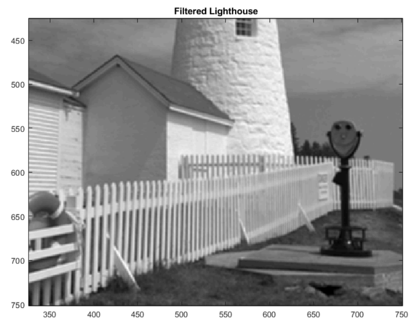
Exercise 4.2

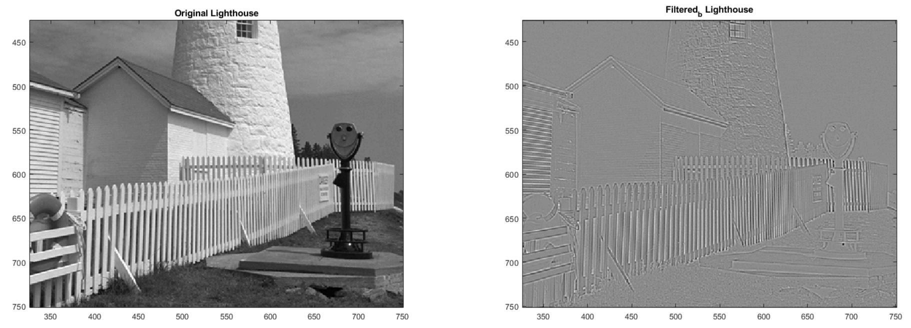
```
%(a)
wa = [1/9,1/9,1/9;1/9,1/9,1/9;1/9,1/9,1/9];
load ('lighthouse.mat');
filtered_a = filter2(wa, lighthouse);

colormap(gray)
figure(1);
subplot(1, 2, 1);
```

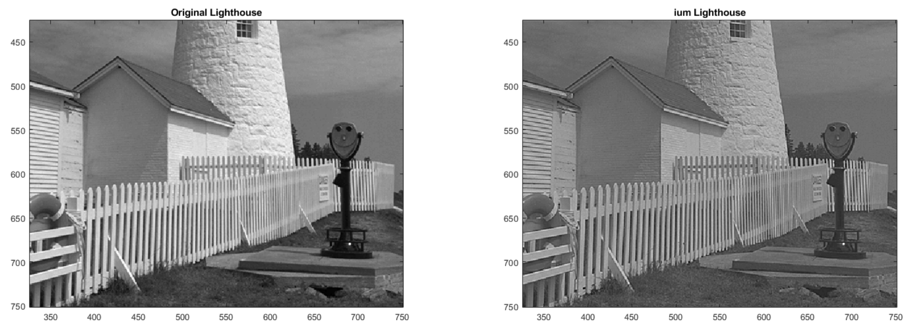
```
imagesc(size(lighthouse,1), size(lighthouse,2), lighthouse);
axis image;
title 'Original Lighthouse';

subplot(1, 2, 2);
imagesc(size(filtered_a,1), size(filtered_a,2), filtered_a);
axis image;
title 'Filtered Lighthouse';
% This blurs the edges in the image.


%(b)
wb = [1/9,1/9,1/9;1/9,-8/9,1/9;1/9,1/9,1/9];
filtered_b = filter2(wb, lighthouse);

colormap(gray)
figure(2);
subplot(1, 2, 1);
imagesc(size(lighthouse,1), size(lighthouse,2), lighthouse);
axis image;
title 'Original Lighthouse';

subplot(1, 2, 2);
imagesc(size(filtered_b,1), size(filtered_b,2), filtered_b);
axis image;
title 'Filtered_b Lighthouse';
% This does a weird effect that sort of makes a lot of the picture
% One shake of gray. The weird/cool thing is, is that you can still
% make out a lot of detail in the picture. It doesn't seem to blur
% the picture, but actually make certain details stand out more.


%(c)
ium = image_unsharp_masking(lighthouse);

colormap(gray)
figure(3);
subplot(1, 2, 1);
imagesc(size(lighthouse,1), size(lighthouse,2), lighthouse);
axis image;
title 'Original Lighthouse';

subplot(1, 2, 2);
imagesc(size(ium,1), size(ium,2), ium);
axis image;
title 'ium Lighthouse';

% This sort of blurs the image, but not really blur it. It seems like
% there is some filter in front of the picture making everything a
% little more faded. The whites are not as white and the blacks are
% note as black.


%(d - (a))
```

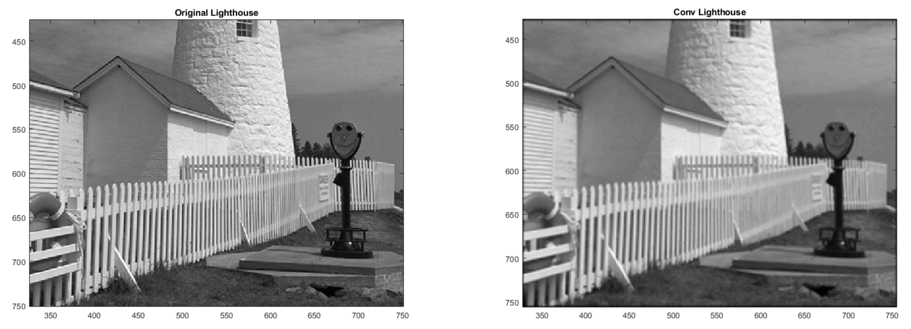
```
conv_a = conv2(wa, lighthouse);

colormap(gray)
figure(4);
subplot(1, 2, 1);
imagesc(size(lighthouse,1), size(lighthouse,2), lighthouse);
axis image;
title 'Original Lighthouse';

subplot(1, 2, 2);
imagesc(size(conv_a,1), size(conv_a,2), conv_a);
axis image;
title 'Conv Lighthouse';
% This seems to do the same thing as filter_a. From using conv and
% filter before, I would expect that the images would be bigger now?
% I don't know why they aren't and I don't know what I am doing wrong.
% I also feel like I am doing something wrong considering it
% specifically asks us to take note of the sizes, but mine stay the
% same. Well really, the conv images is 2 pixels bigger in the x and y
% direction, but that is not a significant difference like I expected.


% (d - (b))
conv_b = conv2(wb, lighthouse);

colormap(gray)
figure(5);
subplot(1, 2, 1);
imagesc(size(lighthouse,1), size(lighthouse,2), lighthouse);
axis image;
title 'Original Lighthouse';

subplot(1, 2, 2);
imagesc(size(conv_b,1), size(conv_b,2), conv_b);
axis image;
title 'Conv_b Lighthouse';
% This seems to do the same thing as filter_b. It also has the same
% size problem as in part (d -(a))


%(d - (c))
ium_conv = image_unsharp_masking_conv(lighthouse);

colormap(gray)
figure(6);
subplot(1, 2, 1);
imagesc(size(lighthouse,1), size(lighthouse,2), lighthouse);
axis image;
title 'Original Lighthouse';

subplot(1, 2, 2);
imagesc(size(ium_conv,1), size(ium_conv,2), ium_conv);
axis image;
title 'ium conv Lighthouse';
```

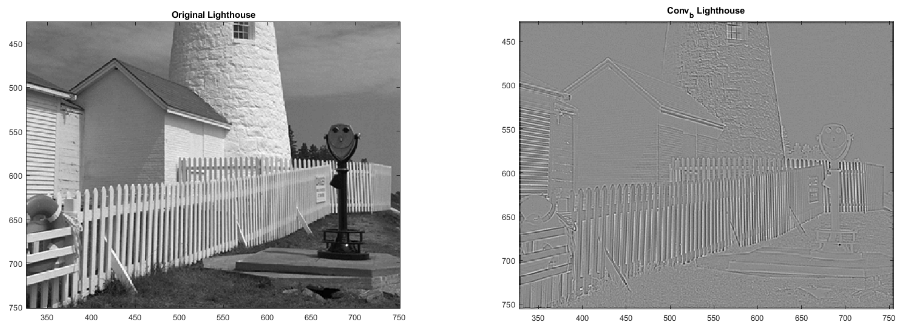
Original Lighthouse
450
500
550
600
650
700
750
350
400
450
500
550
600
650
700
750

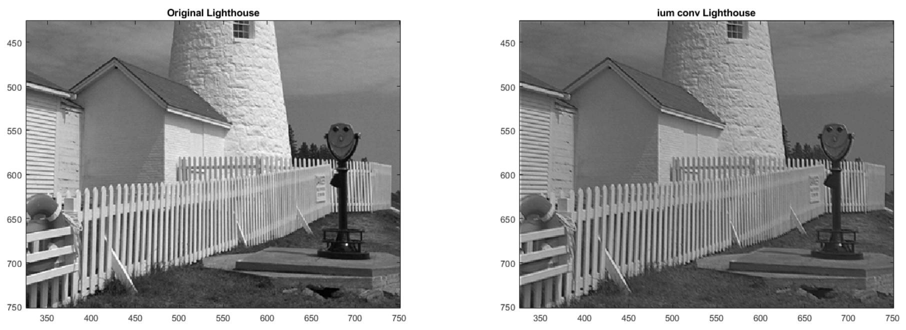

Filtered Lighthouse
450
500
550
600
650
700
750
350
400
450
500
550
600
650
700
750


Original Lighthouse
Filtered$_b$ Lighthouse
450
500
550
600
650
700
750
350
400
450
500
550
600
650
700
750


Original Lighthouse
ium Lighthouse
450
500
550
600
650
700
750
350
400
450
500
550
600
650
700
750


Original Lighthouse
Conv Lighthouse
450
500
550
600
650
700
750
350
400
450
500
550
600
650
700
750





My Functions

```
function y = reverb_conv(x, s, A)
% finds the impulse response h and then convolse the given x with h
a = [1];
b = [1,A];

x_size = size(x,1);
impulse = zeros(1,x_size);
impulse(1) = 1;

h = filter(b,a,impulse);

y = conv(h,x);
```

```
end



function y = reverb_filter(x, s, A)
% finds the impulse response h and uses filter with this impulse
% response and input x
a = [1];
b = [1,A];

x_size = size(x,1);
impulse = zeros(1,x_size);
impulse(1) = 1;

h = filter(b,a,impulse);

y = filter(h,1,x);

end



function y = reverb_own(x, s, A)

x_delayed = [zeros(0,s) x];
x_longer = [x zeros(0,s)];
y = x_longer + A*x_delayed;

end



function y = tremolo(x, fm, A)
% Creates tremolo effect on audio
n = [1 : size(x,1)];
y = x + (A*cos(2*pi*fm*n))*x;

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];
filtered = filter2(wb, im_in);

im_out = im_in - filtered;

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];
```

```
conv = conv2(wb, im_in);

conv(end - 1: end, :) = [];
conv(: ,end - 1 : end) = [];

%I know this is very hacky code, but I didn't know what else to do
%and I was running out of time/had a lot of other classwork to do

im_out = im_in - conv;

end
```

Published with MATLAB® R2020a