ELEC421 – HW4 - Or Bahari 51277200

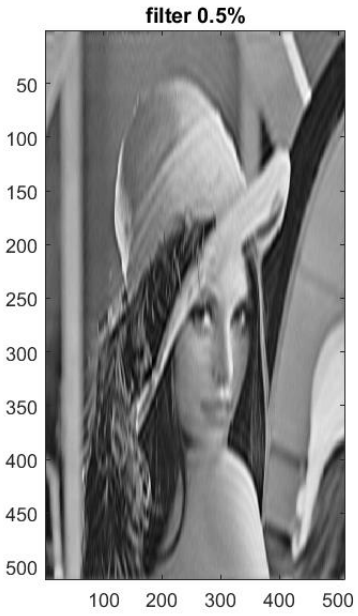
1.

Well basically what I did was to calculate the original power of the signal (in frequency domain) and then play with the value of the radius to get the wanted power percentage. As for an ideal lowpass filter I used a filter that zero-out all the values outside a certain radius from the center of the FFT.

Fliter 0.5%:

perc =

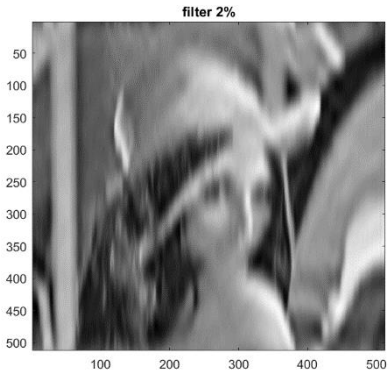
99.5193



Filter 2%:

perc =

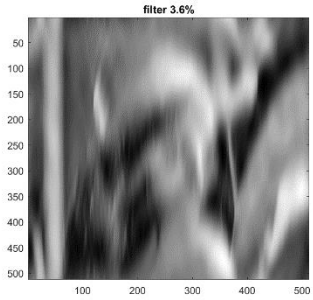
98.2 147



Filter 3.6%:

perc =

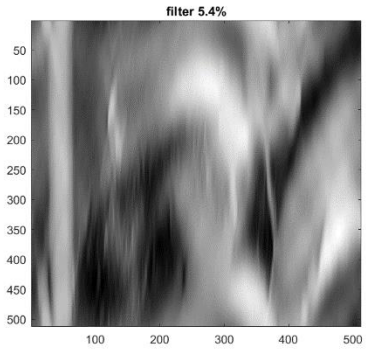
99.4810



Filter 5.4%:

perc =

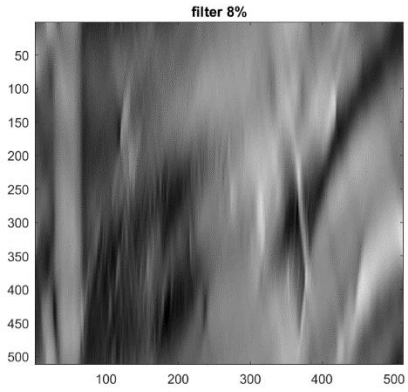
94.8106



Filter 8%:

perc =

91.9327



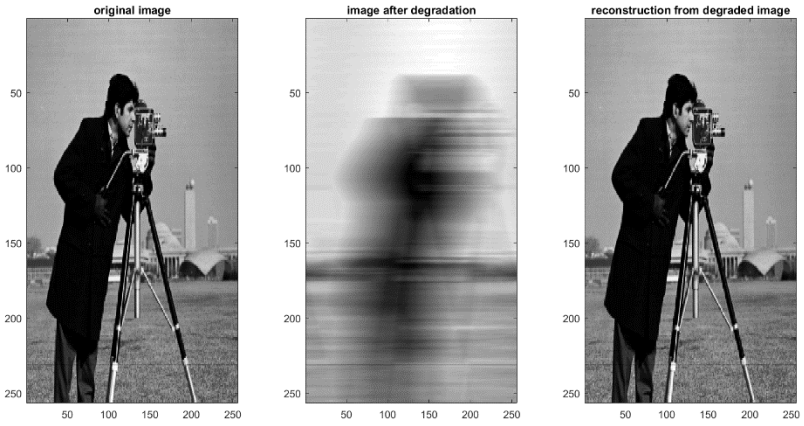
Code:

```
%Q1
l=imread('lena512.bmp');
figure(1);
%subplot(1,2,1);
imshow(l);
title("Lena original image");

L=fftshift((fft2(l)));
rows=size(L,1);
cols=size(L,2);
filter=ones(rows,cols);

orig_power=sum(sum(abs(L).*abs(L)));
rad=1.8; %30 for 0.5%  9 for 2%  4.6 for 3.6%, 3 for 5.4%  1.8 for 8%
for i=1:rows
    for j=1:rows
        if (sqrt((i-rows/2)^2+(j-cols/2))>rad)
            filter(i,j)=0;
        end
    end
end

L=L.*filter;
power=sum(sum(abs(L).*abs(L)));
perc=power/orig_power*100;
restored=ifft2(ifftshift((L)));
%subplot(1,2,2);
imagesc(abs(restored));
title("filter 8%")
colormap 'gray'
```

2.a.

The code creates a filter H in frequency domain that blurs the image (with parameter a) along the x axis. The larger a is the blurrier the filtered image will be. we take the image in the frequency domain and assign H (the fliter) to it. (Then optionally we add a noise `st` to the result), get back to spatial domain and display the blurry image. To reconstruct, we take the inverse filter, 1/H, and applying it to the blurry image in frequency domain. Then we get back to spatial domain and display the image.
Result when running the code with `st`=0:

2.b.



When adding the noise we have something like:

$I_{motion_{fn}} = I \cdot H + N$

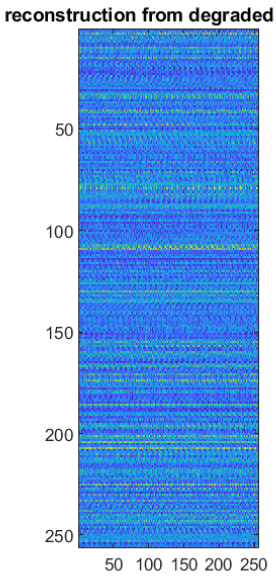
Where N is the noise, H is the filter and I is the image.

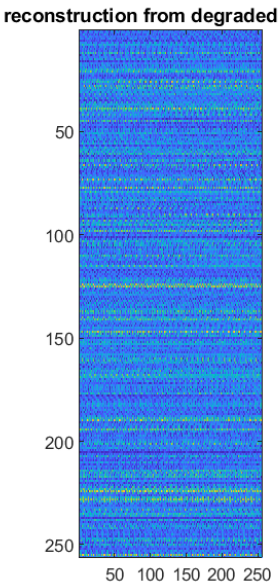
Now, when we try to use the reconstructing filter 1/H, the noise explodes (goes to infinity) and that is why we don't see the reconstructed image (this is the reason we learned about the other sophisticated filters in class like wiener…).

The result is that for both st=0.1 and st=0.001 the reconstructed image is a big mess…

On the right - st=0.01

On the left – st=0.1

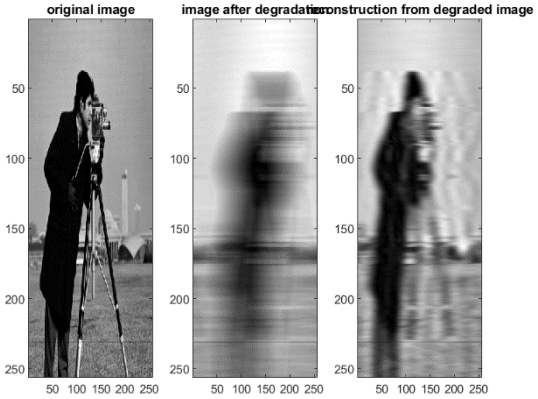




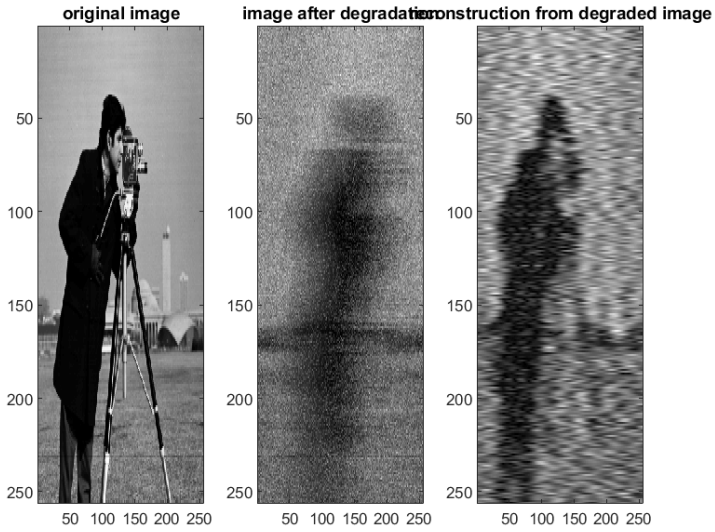
2.c.

What we are doing here is setting a threshold for high values of the inverse filter 1/H (so lower values of H basically). That will make our N/H part in the reconstructed image not to explode.

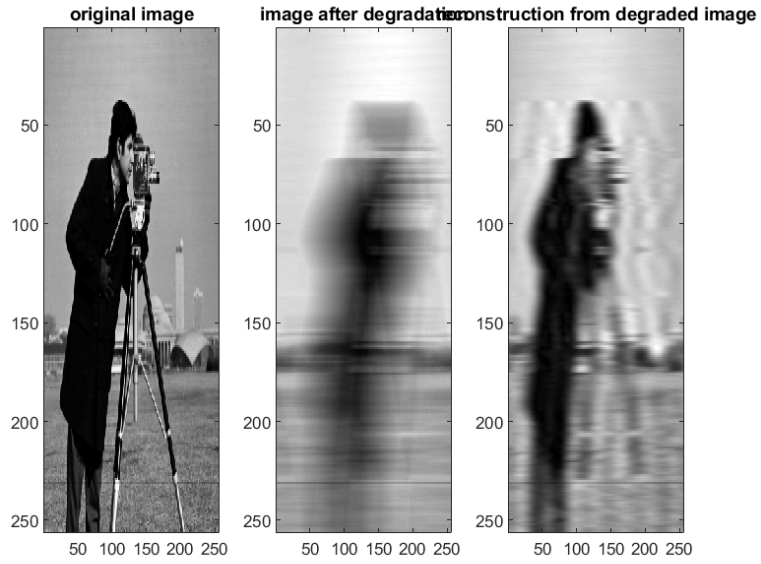
Now we will check the difference between **USE_PSUEDO=1 and USE_PSUEDO=0** (aka question 2.b.) when plugging different `st` values:
St=0 – reconstruction not clear as before (As some parts of H are being taken away…).



st=0.1– reconstruction is better as noise doesn't explode

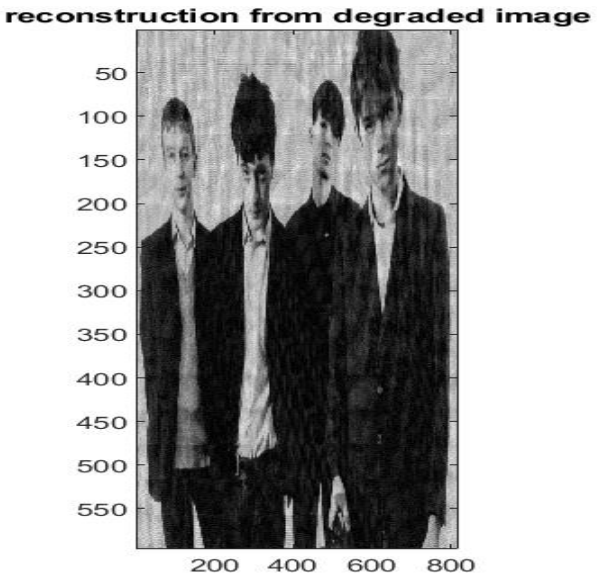


st=0.001– reconstruction is much better with USE_PSUEDO on.



To sum up – as we expected with no noise the normal reconstruction is better but with any noise using a threshold improves reconstruction.

2.d.



Code:

```
mn=0; st=0;  %noise level
USE_PSUEDO=0; % use psuedo filter
%parameter T (observation time) and motion rate
T=1; a=90;ax=30; ay=40;
%reading data
I=im2double(imread('blur.bmp'));
%generating frequencies for the blurring model
u=linspace(-0.5,0.5,size(I,2));
v=linspace(-0.5,0.5,size(I,1));
[U,V]=meshgrid(u,v);
H=(T./(pi*(U*ax+V*ay))).*sin(pi*(U*ax+V*ay)).*exp(-1i*pi*(U*ax+V*ay));
%% Reconstruction
InvFilt = 1./H; %for 2: a-d
if USE_PSUEDO
    threshold = 0.01;
    InvFilt(abs(H)<threshold)=0;
end

%my addition
I_f=fft2(I);
I_motion_fn=fftshift(I_f);
%
I_recon_fn=I_motion_fn.*InvFilt;

I_recon=ifft2(ifftshift(I_recon_fn));
subplot(1,3,3),imagesc(abs(I_recon))
colormap gray;
title('reconstruction from degraded image')
```

2.e.

I used the Wiener filter according to the slides we saw in class:

$$InvFilter = \frac{1}{H} \cdot \frac{|H^2|}{|H^2|+st}$$

Summary – results are better with Wiener filter.

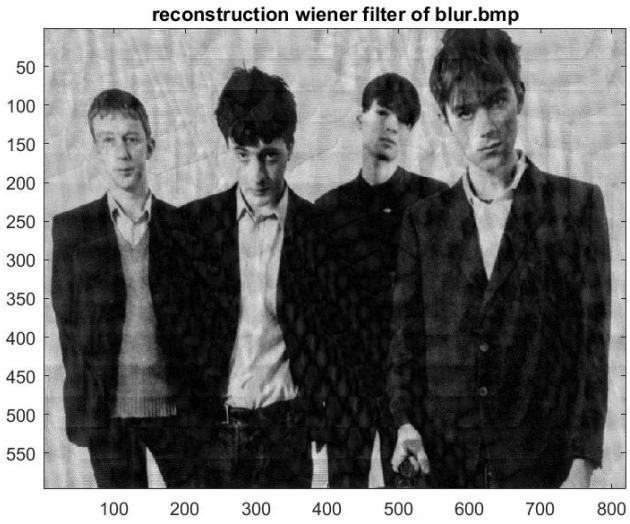
Code:

```
%% Mathematical Model of the Motion Blur
mn=0; st=0;  %0 0.1 0.001

USE_PSUEDO=0; % use psuedo filter

%parameter T (observation time) and motion rate
T=1; a=90;ax=30; ay=40;

%reading data
I=im2double(imread('cameraman.tif'));

%generating frequencies for the blurring model
u=linspace(-0.5,0.5,size(I,2));
v=linspace(-0.5,0.5,size(I,1));

[U,V]=meshgrid(u,v);
H=(T./(pi*U*ax)).*sin(pi*U*ax).*exp(-1i*pi*U*ax);


%% Applying Motion Blur
% filtering in f domain
I_f=fft2(I);
I_motion_f=fftshift(I_f).*H;
```

```
%if you want to add noise to degradation
N=mn+st*randn(size(I));
N_f=fft2(N);
I_motion_fn=I_motion_f+N_f;

figure;
subplot(1,3,1), imagesc(I), colormap(gray)
title('original image')

%back to spatial domain
I_motion_n=ifft2(ifftshift(I_motion_fn));
subplot(1,3,2); imagesc(abs(I_motion_n))
title('image after degradation')

%% Reconstruction
InvFilt= (1./H).*(abs(H).*abs(H))./((abs(H).*abs(H))+st);
I_recon_fn=I_motion_fn.*InvFilt;

I_recon=ifft2(ifftshift(I_recon_fn));
subplot(1,3,3),
imagesc(abs(I_recon))
colormap gray;
title('reconstruction wiener filter st=0')
```

2.f.

I played with the value of st (the parameter of noise in the wiener filter) and found out that we get the clearest image with st=0.0001. (I've **BOLD** it in the code).



Code:

```
I=im2double(imread('blur.bmp'));
%generating frequencies for the blurring model
u=linspace(-0.5,0.5,size(I,2));
v=linspace(-0.5,0.5,size(I,1));

[U,V]=meshgrid(u,v);
H=(T./(pi*(U*ax+V*ay))).*sin(pi*(U*ax+V*ay)).*exp(-1i*pi*(U*ax+V*ay));
InvFilt= (1./H).*(abs(H).*abs(H))./((abs(H).*abs(H))+0.0001 )
I_f=fft2(I);
I_motion_fn=fftshift(I_f);
```

```
I_recon_fn=I_motion_fn.*InvFilt;
I_recon=ifft2(ifftshift(I_recon_fn));
imagesc(abs(I_recon))
colormap gray;
title('reconstruction wiener filter of blur.bmp')
```

3.a.

Code is commented, here are the critical lines:

```
mag = abs(F); %magnitude of Frequency domain of the image

ratio = 0.0103; % The ratio of data we want to notch-out
mask = mag > ratio*max(mag(:)); % we will keep any that satisfies [ratio< mag/MAX(mag)]
mask = bwmorph(mask,'dilate',2); % dilate the binary mask 2 times
mask = bwmorph(mask,'shrink',100); % shrink the image with n=100
mask(size(F,1)/2+1,size(F,2)/2+1) = 0; %0-out the middle value - THE VALUE WE WANT TO KEEP
rad =20; % the radius we want to cut-off each frequency
[x,y] = find(double(mask)); % return the locations of the '1' values in mask
    H = notch('btw',size(F,1),size(F,2),rad,y(i),x(i)); % cuts off frequencies in radius `rad` at
position (x,y)
```

3.b.

Well the notch function zero-out the frequencies of the noise, so we have to make sure the ratio is set such that these frequencies are found properly, then we have to play with the radius of the zeroing out, so we balance between deleting the noise and keeping details of the original image 😊

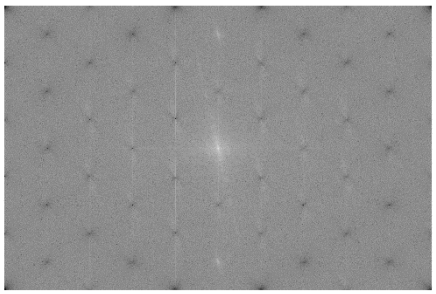
Parameters that worked for me best were:

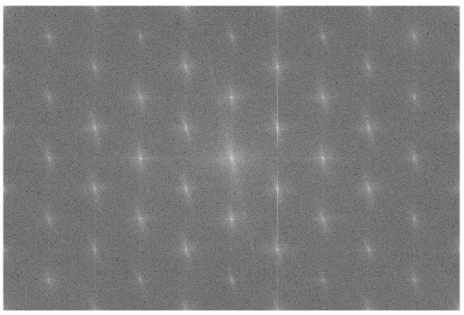
`ratio = 0.0103`

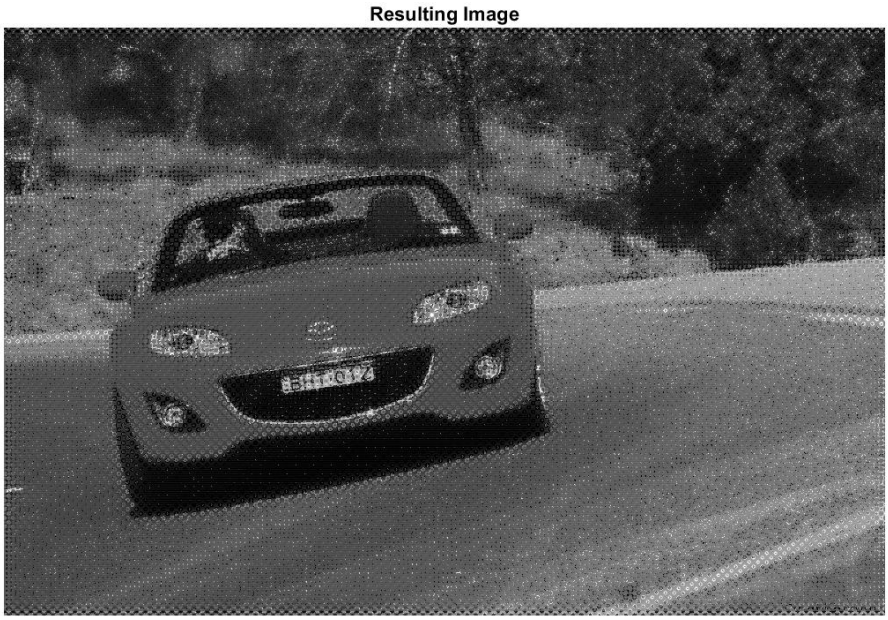
`rad =20`

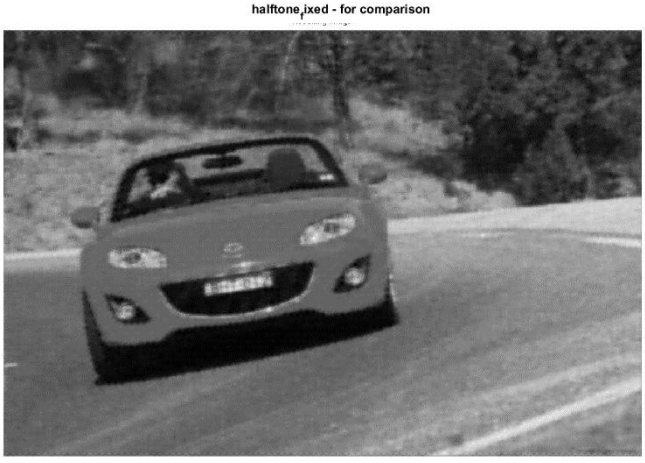
In addition, I chose the notch function to be in mode `**btw**`.

Results: We could see clearly that the plate number is: `**BHT 01Z**` (results are better than halftone_fixed.jpg)



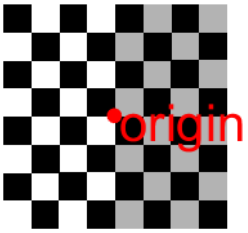


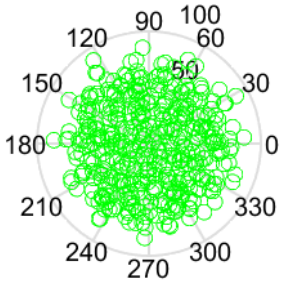




4.a.

Running the code results in:





Code is commented, here are the critical lines:

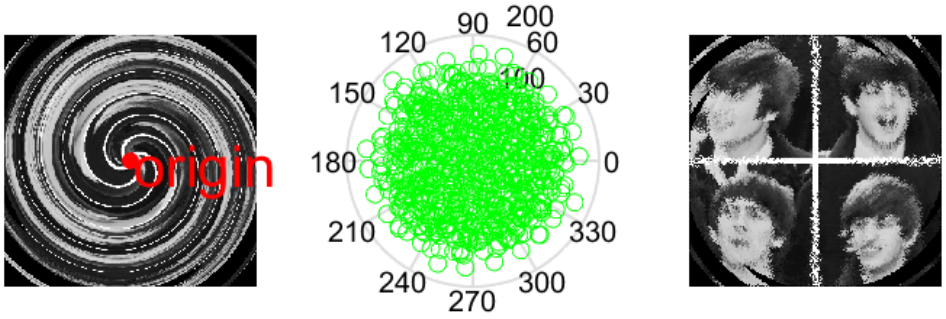
```
pltstp=40; % the distance between each moving point in the polar plot
maxAngRange=0:10:720; %Total spin angle (and how much to rotate every spin)
    for r=1:nRows, %go through all the rows
        for c=1:nCols, %and columns... and then round the numbers  with values between [1<-
>nRows,1<->nCols]
```

4.b.

I just changed the first part of the code:

```
%%
pltstp=100; % the distance between each moving point in the polar plot
%maxAngRange=0:10:720; %Total spin angle (and how much to rotate every spin)
maxAngRange=0:-10:-720;
%A=uint8(255*checkerboard(16));
A=imread('swirled.bmp');
subplot(131); imshow(A);
```

And this is the result:



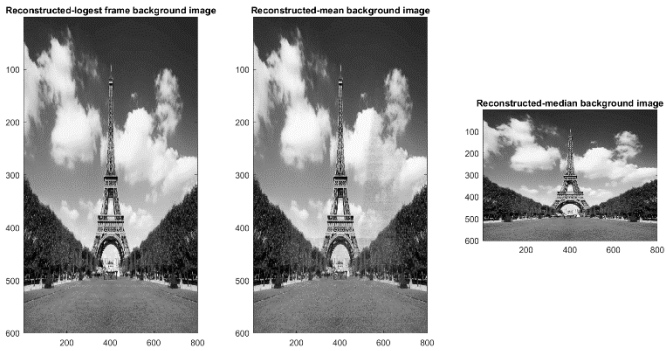
5.a.

Accumulate differences – idea is the track changes between to consecutive pixels and determined the value of the pixel to be the one that was stable for a longer time. In the Paris video it's the values of the background image (And not the moving squares...)

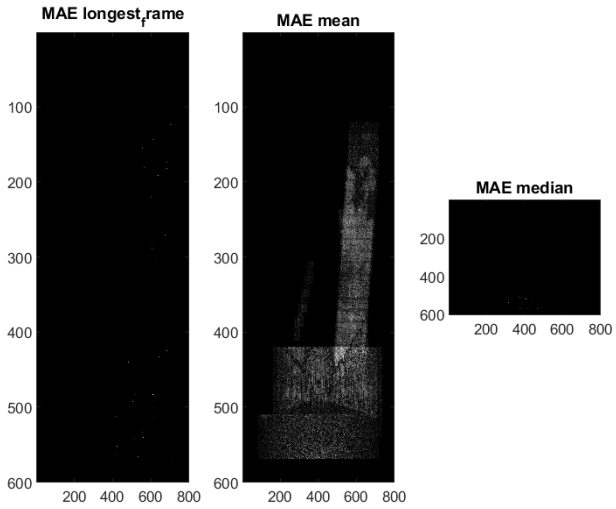
Reconstruct the background image – Keeping the location of the pixels in the video but assigning the values of the ones that were calculated in the Acc-differences part ('the longest-run'/stable pixels).

5.b.

The results are:



And the Mean-Absolute-Error for each method:



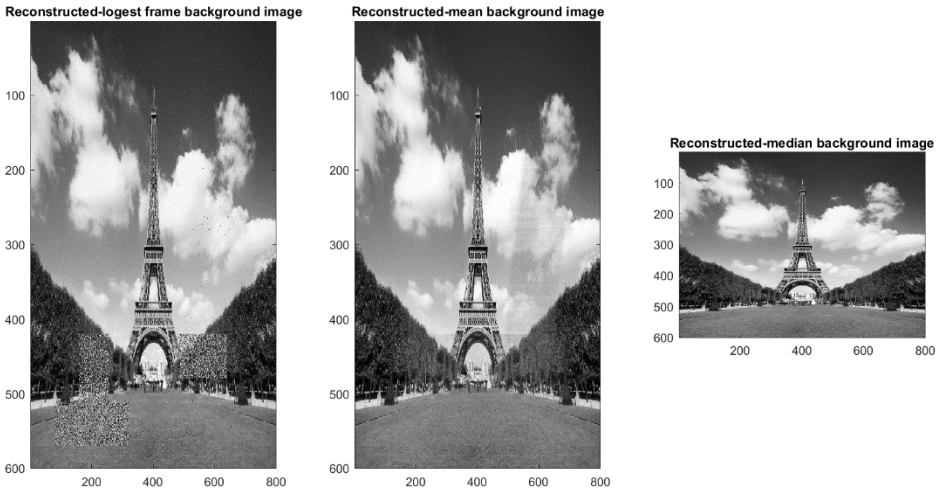
Code (I changed only the `reconstruction` part):

```
%% reconstruct the background image
recon=nan*ones(size(f)); %init same size as one of the frames
for x=1:size(f,1) % for each pixel...
    for y=1:size(f,2) %...
        %set the intensity of recon image at pixel x,y
        %to be intensity of the same pixel location
        %but in the last frame of the longest run
        recon(x,y)=vid(x,y,1,frm_longest_run(x,y));
        rec_mean(x,y)=mean(vid(x,y,1,:)); %just use the average
        rec_med(x,y)=median(vid(x,y,1,:)); %and just use the median
    end
end
figure(51)
subplot(1,3,1)
imagesc(recon)
title("Reconstructed-logest frame background image");

subplot(1,3,2)
imagesc(rec_mean)
title("Reconstructed-mean background image");

subplot(1,3,3)
imagesc(rec_med)
title("Reconstructed-median background image");
```

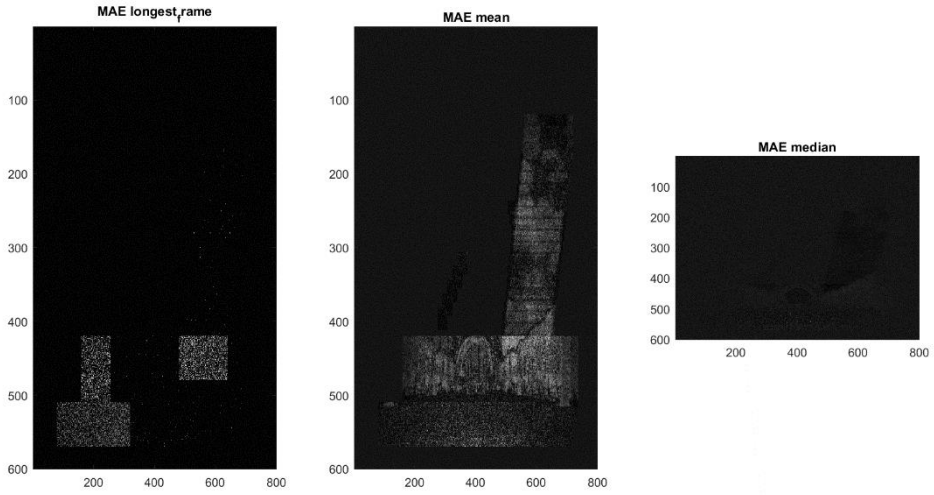
```
axis image
colormap gray

figure(52)
subplot(1,3,1)
imagesc(abs(f-recon))
title('MAE longest_frame')
subplot(1,3,2)
imagesc(abs(f-rec_mean));
title('MAE mean');
subplot(1,3,3)
imagesc(abs(f-rec_med));
axis image
colormap gray
title('MAE median')
```

5.c.

The `nz` parameter adds noise to the background image. With nz=0.05 there is almost no effect on the video. For the reconstruction the results are:



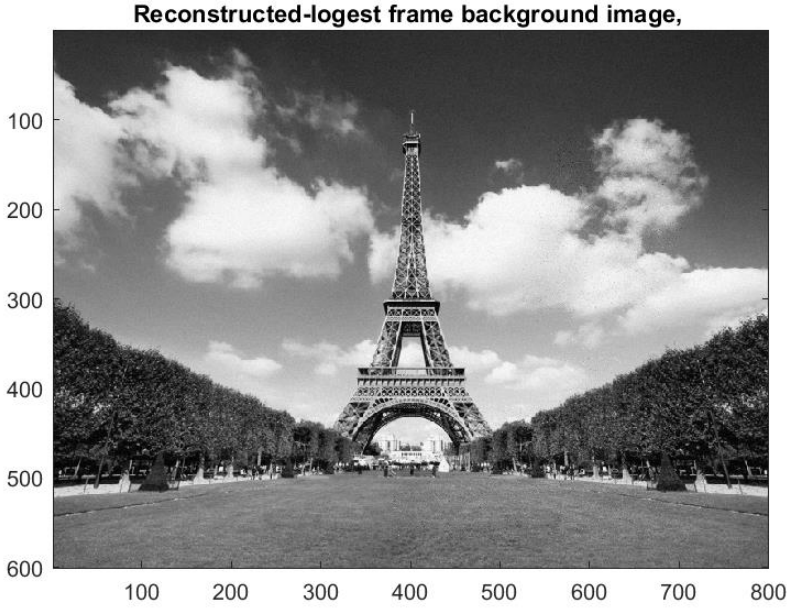
And the MA errors are:



Comment on result – the noise effected the longest_frame method significally and the median has the best results. Fair explanation can be the fact the noise is changing in the image too and the pixels don't really have stable values where the cubes/squares are moving.

5.d.

`T` is a threshold for a real change in the pixel value, so only differences larger than T will pass. With increasing the value of T the lower the noise effects the algorithm. With nz=0.05 and T=0.1 we get:



With error: