Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Assignment 2

DIGITAL IMAGE AND VIDEO PROCESSING

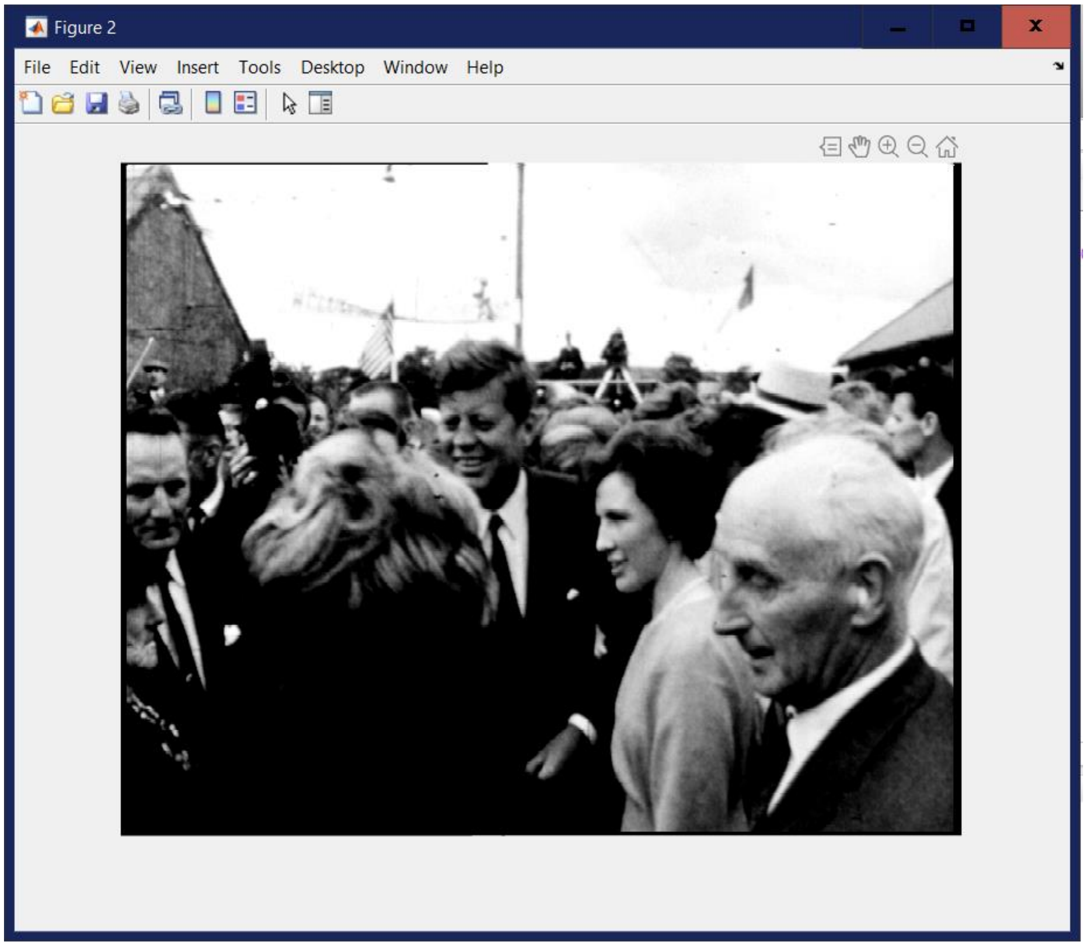
Arnav Malhotra | 17317424 | 26/02/2019

Q2.1. Write an .m script to do a linear contrast stretch on the jfk image from lab 1 so that the image values in the range 55 to 200 are stretched to occupy the whole 8- bit grayscale range. The code will have to ensure that any intensities outside of the range 0:255 are appropriately clipped. Display the output image in a separate figure window when you are done. In your report, include your script along with the image you generate. Explain clearly how the code implements all of the listed requirements.

A2.1.
```
name = imread('D:\Docs\Study\Digital Image and Video
Processing\image_stills\4s1\lab1\Sigmedia06907.tif');
pic = rgb2gray(name);
figure(1);imshow(pic);shg;
t1 = 55;
t2 = 200;

mask = double(pic);
mask = (mask - t1)./(t2-t1);
mask = mask*255;

figure; imshow(uint8(mask));
```



In the code above, after reading the image, it is converted to grayscale. Then, a mask is generated using the values 55 and 200 (as per the question) in order to remove the values from 0 to 55 in the image histogram, change the values of 200 and above to 255 and scale the remaining values to the range of 0 to 255.

Q3.1. Write a matlab function that implements a gaussian low pass filter according to the above criteria. Include the code in your submission and explain what each line of your function does. The function should meet the following criteria.

- The function should take 3 inputs. The first input specifies the variance of the filter mark (positive oat/double), while the second specifies the size (positive integer). The third input is a string which either has the value 'combined' or 'separable'. This parameter is a ag which returns a 1D gaussian filter if the value is separable and a 2D filter if the value is combined.
- There is one output which contains the filter mask/kernel. The size of the output is N*1 (or 1*N) if the separable mode is specified or is N*N if the combined mode is specified. (N is the value of size specified by the 2nd input parameter).
- The function should cater for the possibility of an invalid value of the 3rd parameter. You should define some default behaviour (either return the separable or combined version of the filter mask).
- The maximum value of the filter mask should be at the centre of the array.
- The filter must be symmetric for both odd and even filter lengths.

A3.1.
```
function gaussian = gaussianFilter(sigma, size, type, image)
sigma = input("Enter the sigma:");
size = uint8(size);
size = input("Enter the size:");
type = input("Enter the type:",'s');
sizemod = mod(size,2);
F = zeros(size,size);        %Initialising matrices for filters
H = zeros(1,size);
V = zeros(size,1);
if type == "combined"
    if sizemod == 1                %For odd sized matrices
        for i = 1:size
            for j = 1:size
                F(i,j)=exp(-((ceil(size/2)-i).^2 + (ceil(size/2)-
j).^2)/(2*(sigma^2)));      %Implementation of Gaussian Function
            end
        end
        F = F/sum(F(:));
    else                           %For even sized matrices
        for i = 1:size
            for j = 1:size
                F1(i,j)=exp(-((ceil(size/2)-i)^2 + (ceil(size/2)-
j)^2)/(2*(sigma^2)));       %Implementation of Gaussian Function
            end
        end
        F1 = F1/sum(F1(:));
        for i = 1:size
            for j = 1:size
                F2(i,j)=exp(-((ceil((size/2)+1)-i)^2 +
(ceil(size/2)-j)^2)/(2*(sigma^2)));  %Implementation of Gaussian
Function
            end
        end
        F2 = F2/sum(F2(:));
        for i = 1:size
            for j = 1:size
                F3(i,j)=exp(-((ceil(size/2)-i)^2 +
(ceil((size/2)+1)-j)^2)/(2*(sigma^2)));  %Implementation of
Gaussian Function
            end
        end
        F3 = F3/sum(F3(:));
        for i = 1:size
            for j = 1:size
                F4(i,j)=exp(-((ceil((size/2)+1)-i)^2 +
(ceil((size/2)+1)-j)^2)/(2*(sigma^2)));  %Implementation of
Gaussian Function
            end
        end
        F4 = F4/sum(F4(:));
        F = F1 + F2 + F3 + F4;
        F = F/sum(F(:));            %Normalisation

    end

elseif type == "separable"
    if sizemod == 1                 %For odd sized matrices
        for i = 1:size
            H(i)=exp((-(ceil(size/2)-i)^2)/(2*(sigma^2)));
%Implementation of Gaussian Function
        end
        H = H/sum(H(:));
        for j = 1:size
            V(j)=exp((-(ceil(size/2)-j)^2)/(2*(sigma^2)));
%Implementation of Gaussian Function
        end
        V = V/sum(V(:));
        F = H.*V;                   %Multiplying horizontal and
vertical gaussian filters
        F = F/sum(F(:));            %Normalisation

    else                            %For even sized matrices
        H1 = zeros(1,size);         %Initialising matrices for
filters
```

```
        H2 = zeros(1,size);
        H3 = zeros(1,size);
        H4 = zeros(1,size);
        V1 = zeros(size,1);
        V2 = zeros(size,1);
        V3 = zeros(size,1);
        V4 = zeros(size,1);
        for i = 1:size
            H1(i)=exp((-(ceil(size/2)-i)^2)/(2*(sigma^2)));
%Implementation of Gaussian Function
        end
        H1 = H1/sum(H1(:));
        for j = 1:size
            V1(j)=exp((-(ceil(size/2)-j)^2)/(2*(sigma^2)));
%Implementation of Gaussian Function
        end
        V1 = V1/sum(V1(:));
        F1 = H1.*V1;                   %Multiplying horizontal and
vertical gaussian filters
        F1 = F1/sum(F1(:));            %Normalisation

        for i = 1:size
            H2(i)=exp((-(ceil((size/2)+1)-i)^2)/(2*(sigma^2)));
%Implementation of Gaussian Function
        end
        H2 = H2/sum(H2(:));
        for j = 1:size
            V2(j)=exp((-(ceil(size/2)-j)^2)/(2*(sigma^2)));
%Implementation of Gaussian Function
        end
        V2 = V2/sum(V2(:));
        F2 = H2.*V2;                   %Multiplying horizontal and
vertical gaussian filters
        F2 = F2/sum(F2(:));            %Normalisation

        for i = 1:size
            H3(i)=exp((-(ceil(size/2)-i)^2)/(2*(sigma^2)));
%Implementation of Gaussian Function
        end
        H3 = H3/sum(H3(:));
        for j = 1:size
            V3(j)=exp((-(ceil((size/2)+1)-j)^2)/(2*(sigma^2)));
%Implementation of Gaussian Function
        end
        V3 = V3/sum(V3(:));
        F3 = H3.*V3;                   %Multiplying horizontal and
vertical gaussian filters
        F3 = F3/sum(F3(:));            %Normalisation

        for i = 1:size
```

```
            H4(i)=exp((-(ceil((size/2)+1)-i)^2)/(2*(sigma^2)));
%Implementation of Gaussian Function
        end
        H4 = H4/sum(H4(:));
        for j = 1:size
            V4(j)=exp((-(ceil((size/2)+1)-j)^2)/(2*(sigma^2)));
%Implementation of Gaussian Function
        end
        V4 = V4/sum(V4(:));
        F4 = H4.*V4;                %Multiplying horizontal and
vertical gaussian filters
        F4 = F4/sum(F4(:));         %Normalisation

        F = F1 + F2 + F3 + F4;      %Summing all the filters
        F = F/sum(F(:));            %Normalisation
    end
else                                %Default mode as combined
    if sizemod == 1                 %For odd sized matrices
        for i = 1:size
            for j = 1:size
                F(i,j)=exp(-((ceil(size/2)-i).^2 + (ceil(size/2)-
j).^2)/(2*(sigma^2)));      %Implementation of Gaussian Function
            end
        end
        F = F/sum(F(:));
    else                            %For even sized matrices
        for i = 1:size
            for j = 1:size
                F1(i,j)=exp(-((ceil(size/2)-i)^2 + (ceil(size/2)-
j)^2)/(2*(sigma^2)));       %Implementation of Gaussian Function
            end
        end
        F1 = F1/sum(F1(:));
        for i = 1:size
            for j = 1:size
                F2(i,j)=exp(-((ceil((size/2)+1)-i)^2 +
(ceil(size/2)-j)^2)/(2*(sigma^2)));  %Implementation of Gaussian
Function
            end
        end
        F2 = F2/sum(F2(:));
        for i = 1:size
            for j = 1:size
                F3(i,j)=exp(-((ceil(size/2)-i)^2 +
(ceil((size/2)+1)-j)^2)/(2*(sigma^2)));  %Implementation of
Gaussian Function
            end
        end
        F3 = F3/sum(F3(:));
        for i = 1:size
```

```
            for j = 1:size
                F4(i,j)=exp(-((ceil((size/2)+1)-i)^2 +
(ceil((size/2)+1)-j)^2)/(2*(sigma^2)));  %Implementation of
Gaussian Function
            end
        end
        F4 = F4/sum(F4(:));
        F = F1 + F2 + F3 + F4;
        F = F/sum(F(:));            %Normalisation

    end
end

figure(1),imshow(image);
image = imfilter(image, F);         %Convolving image and the
final Gaussian Filter
figure(2),imshow(image)

end
```

Applying gaussianFilter-

```
I = imread('D:\Docs\Study\Digital Image and Video
Processing\image_stills\4s1\lab1\Sigmedia06907.tif');
image = rgb2gray(I);
figure(1),imshow(image)
image = gaussianFilter(0,0,0,image);
figure(2),imshow(image)
```

After receiving user input regarding sigma, size and type (combined or separable), which loop would be running is decided. This was done as there are 4 cases which need to be dealt with:

1) Type = combined, size is odd.
2) Type = combined, size is even.
3) Type = separable, size is odd.
4) Type = separable, size is even.

Each of them is applied by using these formulae and subsequently performing normalisation:

$$f[h,k] = \exp\left(-\frac{h^2+k^2}{2\sigma^2}\right) \qquad f_1[h] = \exp\left(-\frac{h^2}{2\sigma^2}\right)$$

$$f_2[k] = \exp\left(-\frac{k^2}{2\sigma^2}\right).$$

After the filter is calculated, it is convolved with the matrix of the image giving us the blurred image.

Q3.2. Explain why it is necessary that all the coefficients of the low pass filter sum to 1.

A3.2. If the sum is more than one, then it would result in the amplification of magnitude of the pixels making the image light. On the other hand, if the sum is less than one, then attenuation of magnitude would take place resulting in a dark image.

Q3.3. Changing the value of N to 21 and using the matlab functions tic and toc, record the time that it takes to apply the system to the image 1000 times. The filter should be applied to both the rows and the columns of the image. In your answer include the code you use and write down the time taken.

A3.2. Applying the above conditions, Separable Gaussian Filter takes 14.434505 seconds.
Applying gaussianFilterSep-

```
I = imread('D:\Docs\Study\Digital Image and Video
Processing\image_stills\4s1\lab1\Sigmedia06907.tif');
image = rgb2gray(I);
figure(1),imshow(image);
tic
for i = 1:1000
    image = gaussianFilterSep(1,21,"separable",image);
end
toc
t = toc;
figure(2),imshow(image);
```

Function-

```
function image = gaussianFilterSep(sigma, size, type, image)
%sigma = input("Enter the sigma:");
%size = uint8(size);
%size = input("Enter the size:");
%type = input("Enter the type:",'s');
sizemod = mod(size,2);
F = zeros(size,size);       %Initialising matrices for filters
H = zeros(1,size);
V = zeros(size,1);
```

```
if sizemod == 1                  %For odd sized matrices
        for i = 1:size
            H(i)=exp((-(ceil(size/2)-i)^2)/(2*(sigma^2)));
%Implementation of Gaussian Function
        end
        H = H/sum(H(:));
        for j = 1:size
            V(j)=exp((-(ceil(size/2)-j)^2)/(2*(sigma^2)));
%Implementation of Gaussian Function
        end
        V = V/sum(V(:));
        F = H.*V;                    %Multiplying horizontal and
vertical gaussian filters
        F = F/sum(F(:));             %Normalisation

else                             %For even sized matrices
        H1 = zeros(1,size);          %Initialising matrices for
filters
        H2 = zeros(1,size);
        H3 = zeros(1,size);
        H4 = zeros(1,size);
        V1 = zeros(size,1);
        V2 = zeros(size,1);
        V3 = zeros(size,1);
        V4 = zeros(size,1);
        for i = 1:size
            H1(i)=exp((-(ceil(size/2)-i)^2)/(2*(sigma^2)));
%Implementation of Gaussian Function
        end
        H1 = H1/sum(H1(:));
        for j = 1:size
            V1(j)=exp((-(ceil(size/2)-j)^2)/(2*(sigma^2)));
%Implementation of Gaussian Function
        end
        V1 = V1/sum(V1(:));
        F1 = H1.*V1;                 %Multiplying horizontal and
vertical gaussian filters
        F1 = F1/sum(F1(:));          %Normalisation

        for i = 1:size
            H2(i)=exp((-(ceil((size/2)+1)-i)^2)/(2*(sigma^2)));
%Implementation of Gaussian Function
        end
        H2 = H2/sum(H2(:));
        for j = 1:size
            V2(j)=exp((-(ceil(size/2)-j)^2)/(2*(sigma^2)));
%Implementation of Gaussian Function
        end
        V2 = V2/sum(V2(:));
        F2 = H2.*V2;                    %Multiplying horizontal and
vertical gaussian filters
        F2 = F2/sum(F2(:));             %Normalisation

        for i = 1:size
            H3(i)=exp((-(ceil(size/2)-i)^2)/(2*(sigma^2)));
%Implementation of Gaussian Function
        end
        H3 = H3/sum(H3(:));
        for j = 1:size
            V3(j)=exp((-(ceil((size/2)+1)-j)^2)/(2*(sigma^2)));
%Implementation of Gaussian Function
        end
        V3 = V3/sum(V3(:));
        F3 = H3.*V3;                    %Multiplying horizontal and
vertical gaussian filters
        F3 = F3/sum(F3(:));             %Normalisation

        for i = 1:size
            H4(i)=exp((-(ceil((size/2)+1)-i)^2)/(2*(sigma^2)));
%Implementation of Gaussian Function
        end
        H4 = H4/sum(H4(:));
        for j = 1:size
            V4(j)=exp((-(ceil((size/2)+1)-j)^2)/(2*(sigma^2)));
%Implementation of Gaussian Function
        end
        V4 = V4/sum(V4(:));
        F4 = H4.*V4;                    %Multiplying horizontal and
vertical gaussian filters
        F4 = F4/sum(F4(:));             %Normalisation

        F = F1 + F2 + F3 + F4;          %Summing all the filters
        F = F/sum(F(:));                %Normalisation
end


%figure(1),imshow(image);
image = imfilter(image, F);             %Convolving image and the
final Gaussian Filter
%figure(2),imshow(image)

end
```

Q3.4. Change your m-file to use the full 2D implementation of the filter (ie. not separably). Include your code and write down the time it takes to apply this separable implementation 1000 times as before.

A3.4. Applying the above conditions, Combined Gaussian Filter takes 15.550710 seconds.
Applying gaussianFilterComb-

```
I = imread('D:\Docs\Study\Digital Image and Video
Processing\image_stills\4s1\lab1\Sigmedia06907.tif');
image = rgb2gray(I);
figure(1),imshow(image);
tic
for i = 1:1000
    image = gaussianFilterComb(1,21,"combined",image);
end
toc
t = toc;
figure(2),imshow(image);
```

Function-

```
function image = gaussianFilterComb(sigma, size, type, image)
%sigma = input("Enter the sigma:");
%size = uint8(size);
%size = input("Enter the size:");
%type = input("Enter the type:",'s');
sizemod = mod(size,2);
F = zeros(size,size);       %Initialising matrices for filters

if sizemod == 1              %For odd sized matrices
        for i = 1:size
            for j = 1:size
                F(i,j)=exp(-((ceil(size/2)-i).^2 + (ceil(size/2)-
j).^2)/(2*(sigma^2)));     %Implementation of Gaussian Function
            end
        end
        F = F/sum(F(:));
    else                             %For even sized matrices
        for i = 1:size
            for j = 1:size
                F1(i,j)=exp(-((ceil(size/2)-i)^2 + (ceil(size/2)-
j)^2)/(2*(sigma^2)));      %Implementation of Gaussian Function
            end
        end
        F1 = F1/sum(F1(:));
        for i = 1:size
```

```
            for j = 1:size
                F2(i,j)=exp(-((ceil((size/2)+1)-i)^2 +
(ceil(size/2)-j)^2)/(2*(sigma^2)));  %Implementation of Gaussian
Function
            end
        end
        F2 = F2/sum(F2(:));
        for i = 1:size
            for j = 1:size
                F3(i,j)=exp(-((ceil(size/2)-i)^2 +
(ceil((size/2)+1)-j)^2)/(2*(sigma^2)));  %Implementation of
Gaussian Function
            end
        end
        F3 = F3/sum(F3(:));
        for i = 1:size
            for j = 1:size
                F4(i,j)=exp(-((ceil((size/2)+1)-i)^2 +
(ceil((size/2)+1)-j)^2)/(2*(sigma^2)));  %Implementation of
Gaussian Function
            end
        end
        F4 = F4/sum(F4(:));
        F = F1 + F2 + F3 + F4;
        F = F/sum(F(:));            %Normalisation

end


%figure(1),imshow(image);
image = imfilter(image, F);         %Convolving image and the
final Gaussian Filter
%figure(2),imshow(image)

End
```

Q3.5. Is there any difference in execution time between these two implementations? Explain your findings.

A3.5. Yes, there is a difference in execution time of about 1.116205 seconds where separable mode has proven to be faster than combined mode. This is because using a two 1-D filters (separable) instead of one 2-D filter (combined) is computationally less expensive and proves to be very useful on significantly larger scales.

Q4.1. Write an m-file that loads up the sigmedia06907.tif image and extracts the three colour planes. Then using the Gaussian filter above as a low pass filter, implement the system in the figure below and apply it to each channel of the input image. The Gaussian filter (Low Pass Filter) has user defined variance sigma^2 and N taps. The fraction of high pass information that is added back into the image is f. Use N = 15 and sigma^2 = 2.5^2. Along with your code the answer should explain how the unsharp masking filter is implemented. The code that you submit for this section will be partially assessed on its neatness, clarity and computational efficiency.

A4.1.
```
I = imread('D:\Docs\Study\Digital Image and Video
Processing\image_stills\4s1\lab1\Sigmedia06907.tif');

red_plane = I(:,:,1);
[red] = gaussianFilter2(2.5,15,"combined",red_plane);
%Applying Gaussian Filter on red channel

green_plane = I(:,:,2);
[green] = gaussianFilter2(2.5,15,"combined",green_plane);
%Applying Gaussian Filter on red channel

blue_plane = I(:,:,3);
[blue] = gaussianFilter2(2.5,15,"combined",blue_plane);
%Applying Gaussian Filter on red channel

[low_pass] = cat(3, red, green, blue);
%Combining all channels
figure(1),imshow(I)
%figure(9),imshow(low_pass)
high_pass = I - low_pass;
%Calculating the high pass filter fraction of which would be used
f = 0.3;
high_pass = f*high_pass;
final = I + high_pass;
figure(2),imshow(final)
```

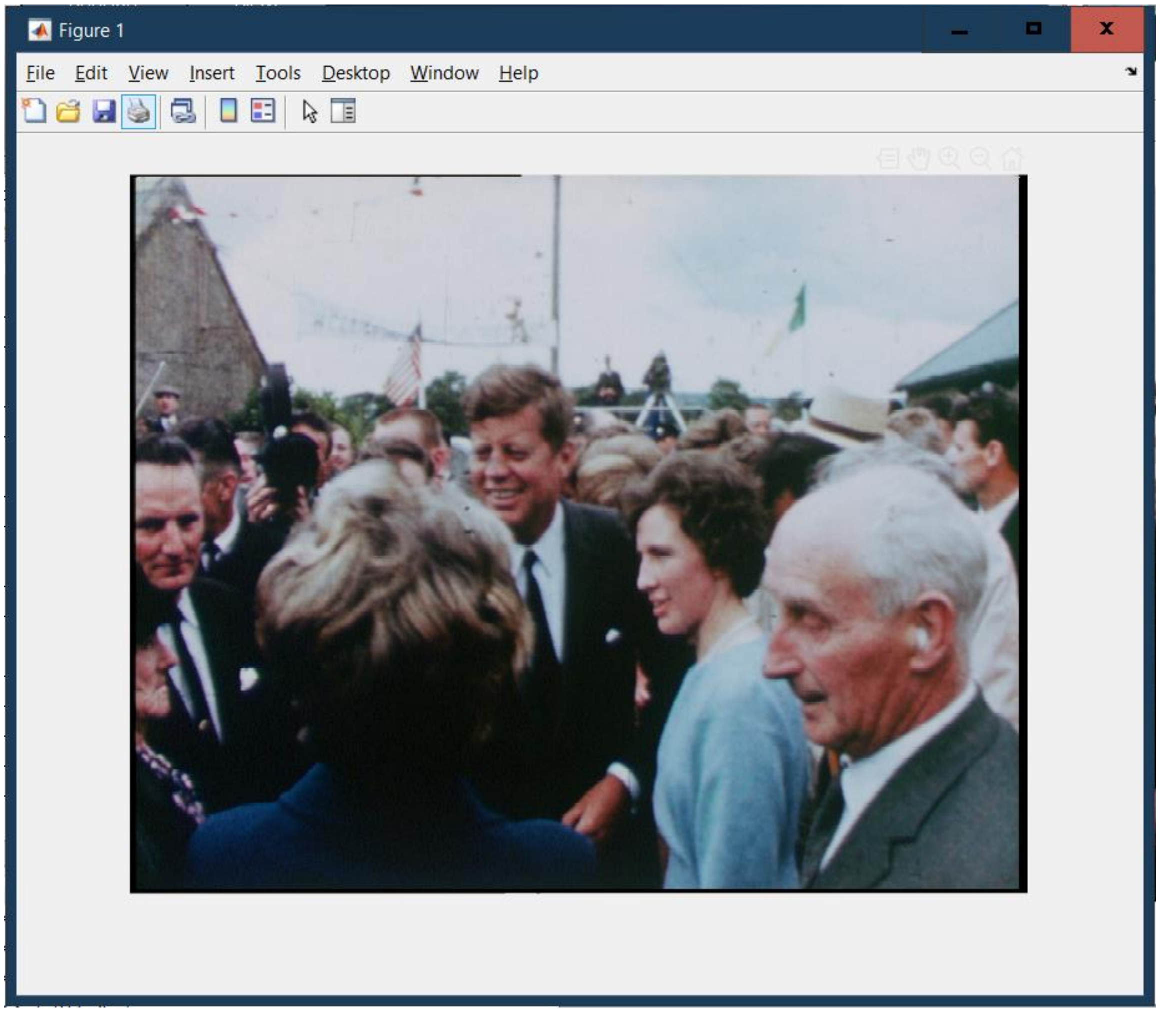
After computing the filters for each channel, they are combined to get the low pass filter. This is in turn subtracted from the input image in order to get the high pass filter fraction of which is combined with the input image to get the final output.

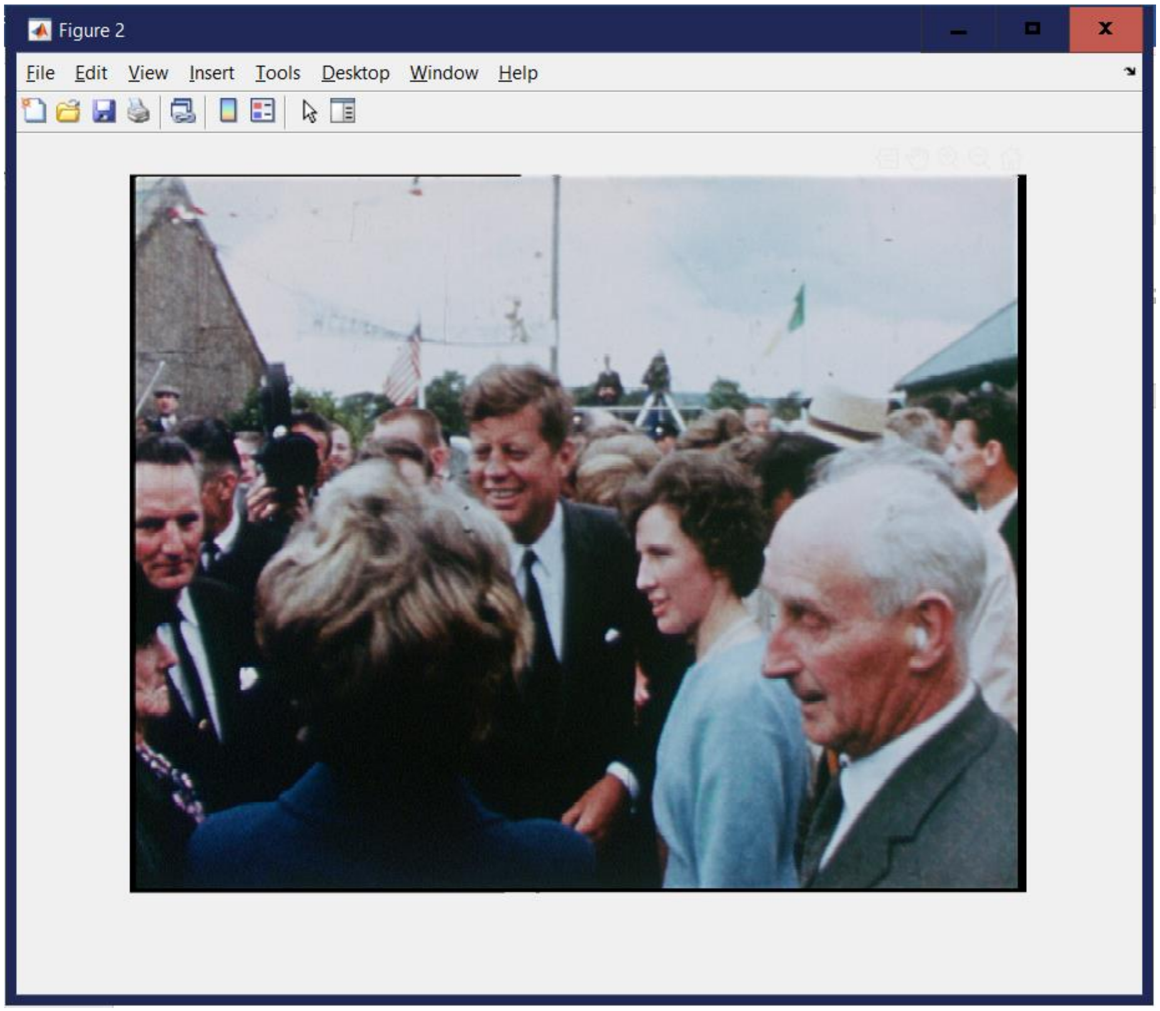
Q4.2. By applying the system and observing the output image in colour, pick a value of f that gives the best-looking result. Write the value of f.

A4.2. The value of f suitable for this operation is 0.3.

Q4.3. Describe the appearance of your output compared with the input.

A4.3. Comparing the images, it can be seen that the latter image is considerably sharper than the former.





Q4.4. Justify the use of the selected value.

A4.4. f = 0.3 has been taken as higher values introduced artefacts into the image (such as aliasing).

Q4.5. Now process the pool.01.bmp image with these same settings. Is the result as good? Explain your findings.

A4.5. For lower values of f, image is sharpened. On the other hand, there is a large number of artefacts which are introduced due to f being higher. Hence, for f = 0.15, the results are better.

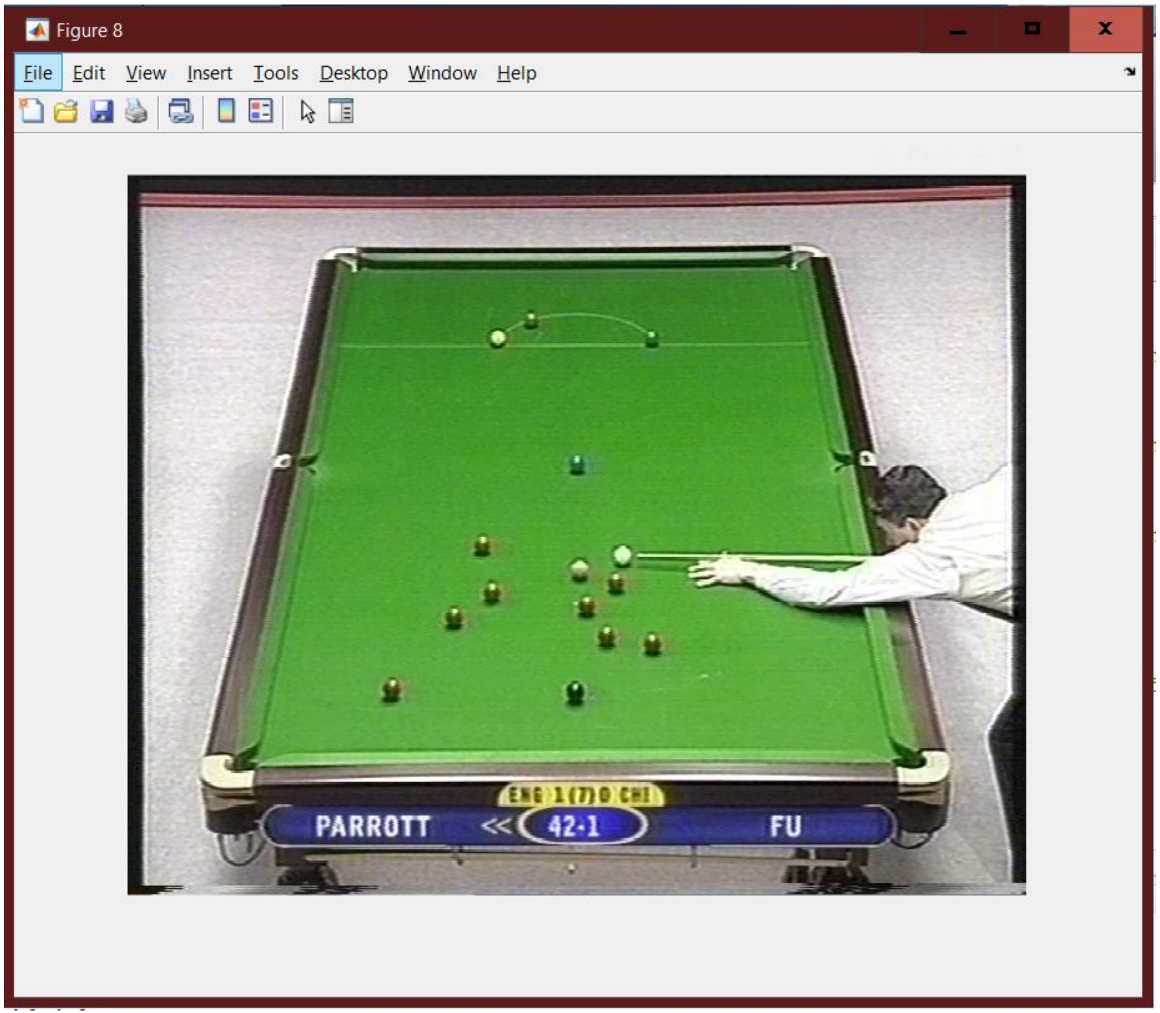


Figure 8
File Edit View Insert Tools Desktop Window Help
ENG 1 (7) 0 CHI
PARROTT
42-1
FU

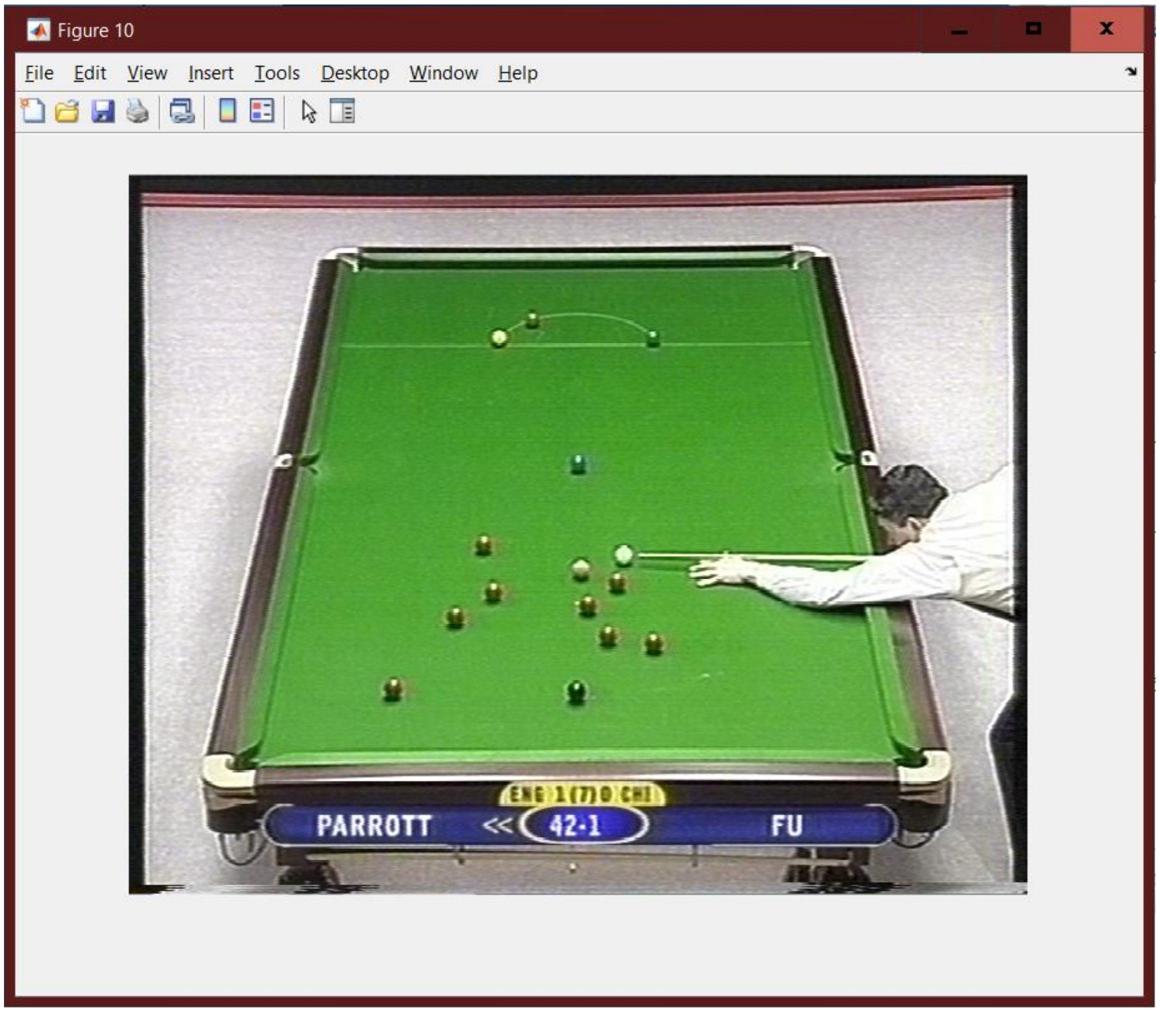


Figure 10
File Edit View Insert Tools Desktop Window Help
ENG 1 (7) 0 CHI
PARROTT
42-1
FU