q1

January 20, 2018

1 Resizing image Using Interpolation

1.1 Importing images

```
In [5]: img1 = imread('./blur.jpg');
```

```
In [21]: img2 = imread('./cameraman.png');
```

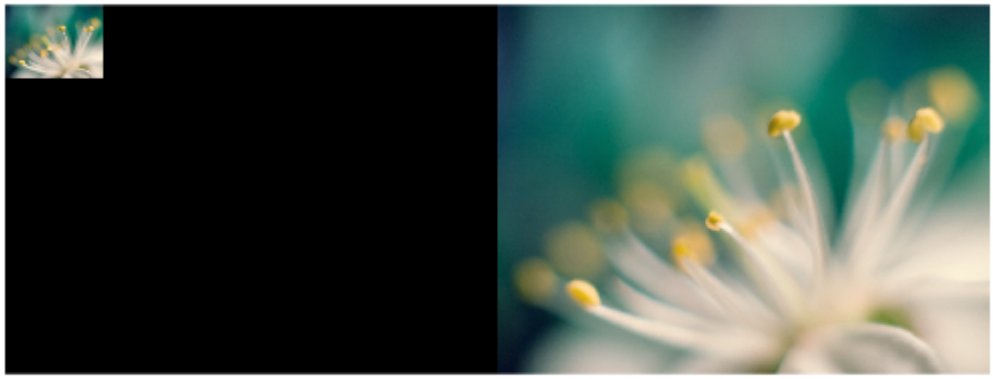
```
In [8]: img3 = imread('./shapes.gif');
```

1.2 Using Neighrest Neighbours

In nearest neighbour the closest pixel values is copied to the pixel which results in a more pixelated image which can be used accordingly in particular types of art

```
In [12]: new_img1 = RESIZENN(img1,5);
         imshowpair(img1,new_img1,'montage');
```

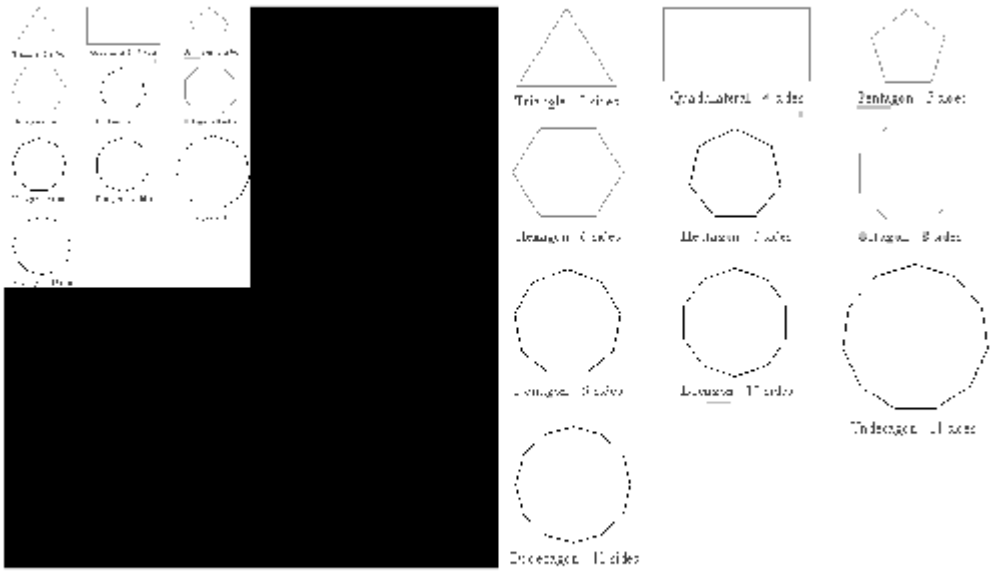
```
Warning: Image is too big to fit on screen; displaying at 8%
> In images.internal.initSize (line 71)
  In imshow (line 328)
  In imshowpair (line 126)
```



```
In [23]: new_img2= RESIZENN(img2,2);
         imshowpair(img2,new_img2,'montage');
```

```
In [22]: new_img3= RESIZENN(img3,2);
         imshowpair(img3,new_img3,'montage');
```

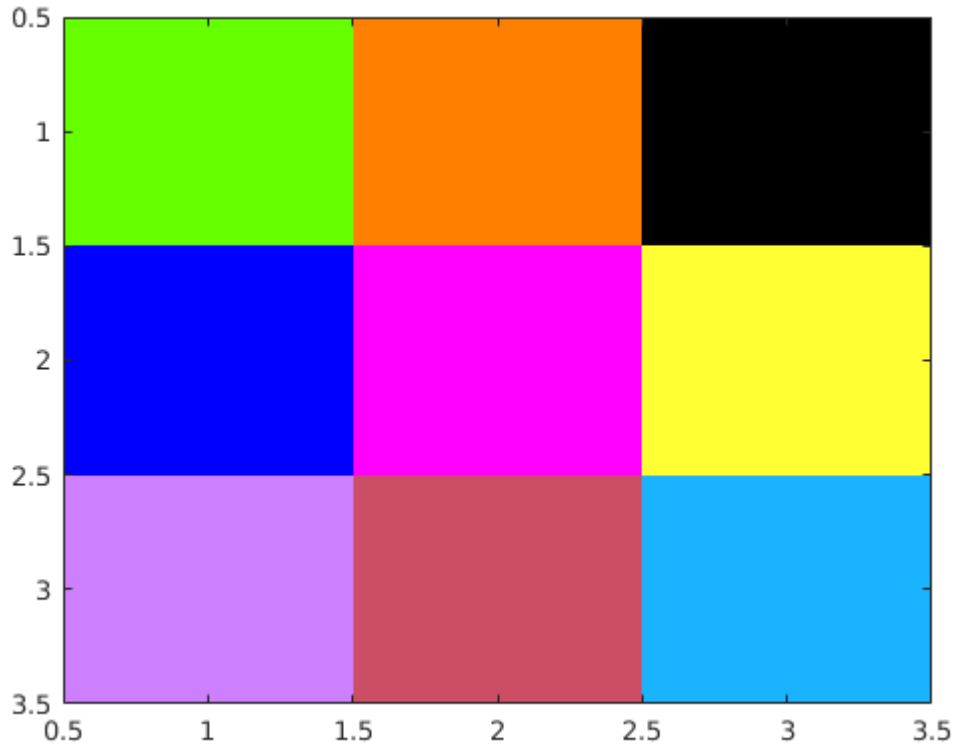
```
Warning: Image is too big to fit on screen; displaying at 67%
> In images.internal.initSize (line 71)
  In imshow (line 328)
  In imshowpair (line 126)
```



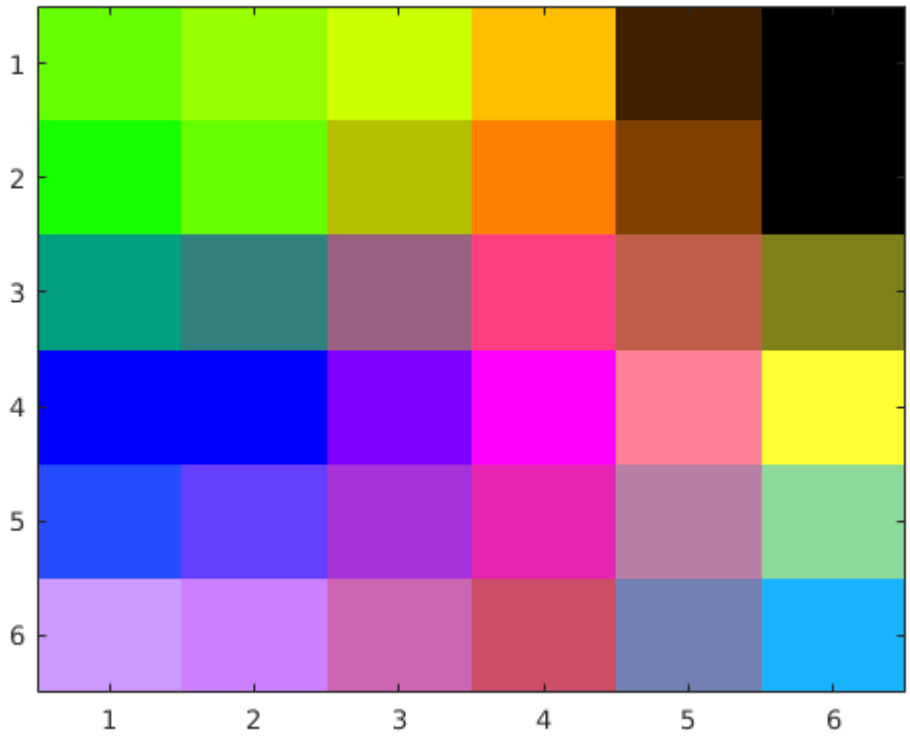
2 Using Bilinear Interpolation

Bilinear interpolation results in a smoother image due to rowise linear interpolation i.e the contribution of the closer pixel is more than the farther one but still both contribute this results in a smoother picture.

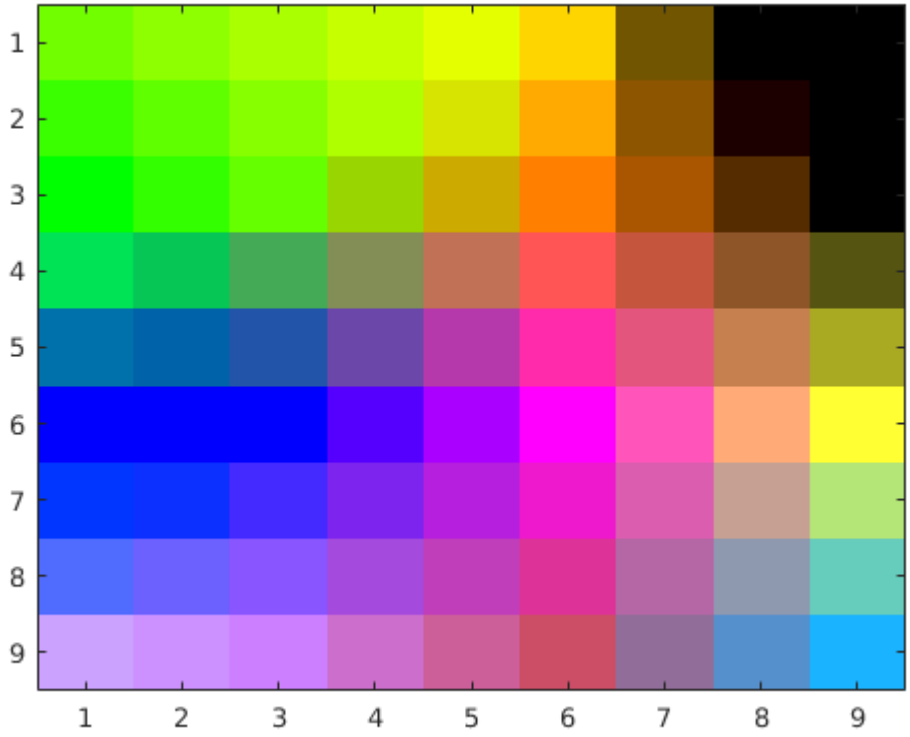
```
In [25]: red = [0.4 1 0; 0 1 1; 0.8 0.8 0.1 ];
         green = [1 0.5 0; 0 0 1; 0.5 0.3 0.7];
         blue = [0.0 0.0 0; 1.0 1.0 0.2; 1.0 0.4 1];
         img = cat(3, red, green, blue);
         image(img);
```




```
In [32]: new_img = RESIZEBL(img,2);
         image(new_img)
```

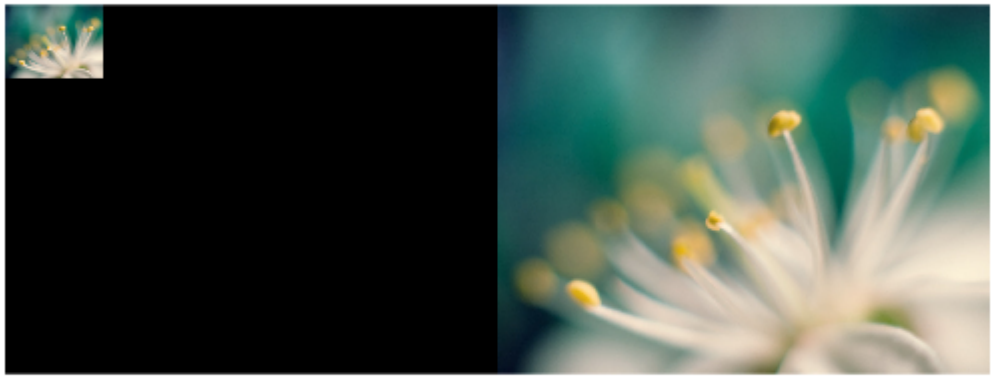



```
In [33]: new_img = RESIZEBL(img,3);
         image(new_img)
```




```
In [34]: new_img1 = RESIZEBL(img1,5);
         imshowpair(img1,new_img1,'montage');
```

```
Warning: Image is too big to fit on screen; displaying at 8%
> In images.internal.initSize (line 71)
  In imshow (line 328)
  In imshowpair (line 126)
```



```
In [35]: new_img2= RESIZEBL(img2,2);
         imshowpair(img2,new_img2,'montage');
```

2.0.1 The Difference between the two types of interpolations

The major difference between the two is that the images resized be nearest neighbpur are more pixelated and those done by bilear are not. Nearest neighbour fairs equal or even better with images with higher number of straight edges

We have a lot of other alternatives for image resizing like bicubic interpolation which uses the cubic equations for finding the pixel values.

In [ ]:

q2

January 20, 2018

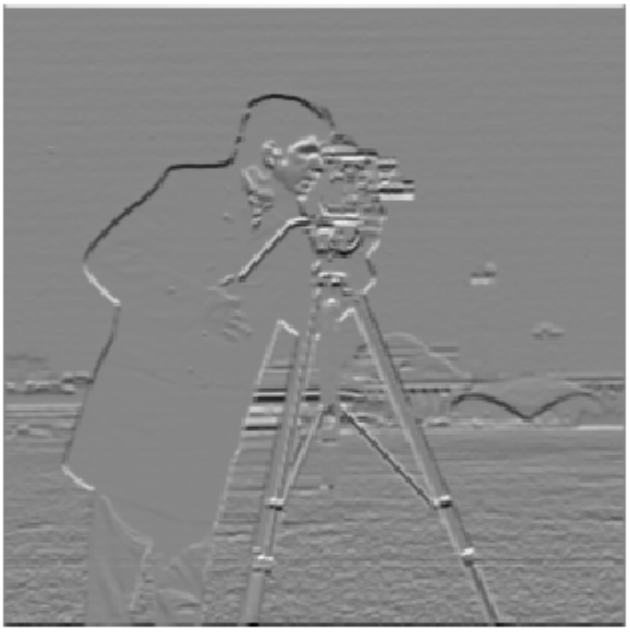
1 Applying and implementing convolution

In [3]:
```
img = imread('./cameraman.png');
imshow(img)
size(img);
```

In [6]:
```
M = [1 2 1; 0 0 0; -1 -2 -1];
M = double(M);
```

2 Detecting Horizontal Edges

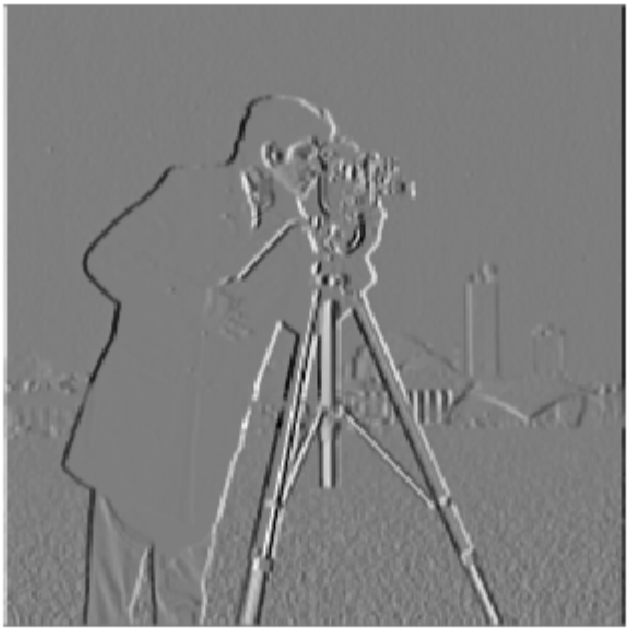
In [9]:
```
new_img = conv2(M,img);
new_img = mat2gray(new_img);
imshow(new_img)
```



3 Detecting Vertical Edges

By taking the transpose we can find Vertical images

```
In [10]: new_img = conv2(M',img);
         new_img = mat2gray(new_img);
         imshow(new_img)
```



Originally we can't see much difference but after converting it to gray we can see the edges detected

In [ ]:

q3

January 20, 2018

1 Outsize after convolution

We are given - Image size = Width, Height, Channels - No. of filters = N - filter size = F,F,Channels - Stride = S - padding = Z

Output Size = N(Width - F + 2P)/S + 1, N(Height -F +2P)/S + 1, Channels)

2 No. of Muliplications:

Multiplications in 1 Operation = FF Addition in 1 Operation = 1 Total Operations = Output_widthOutput_Height*Channels

Total Multiplication = (FF) (Output_widthOutput_HeightChannels) Total Addition = 1 * Output_widthOutput_HeightChannels

Reference

In [ ]:

q4

January 20, 2018

1 Audio Convolution

```
In [10]: audio_file = audiorecorder(44100,24,1,1)
```

```
audio_file =

  audiorecorder with properties:

         SampleRate: 44100
      BitsPerSample: 24
    NumberOfChannels: 1
           DeviceID: 1
      CurrentSample: 1
       TotalSamples: 0
            Running: 'off'
           StartFcn: []
            StopFcn: []
           TimerFcn: []
        TimerPeriod: 0.0500
                Tag: ''
           UserData: []
               Type: 'audiorecorder'
```

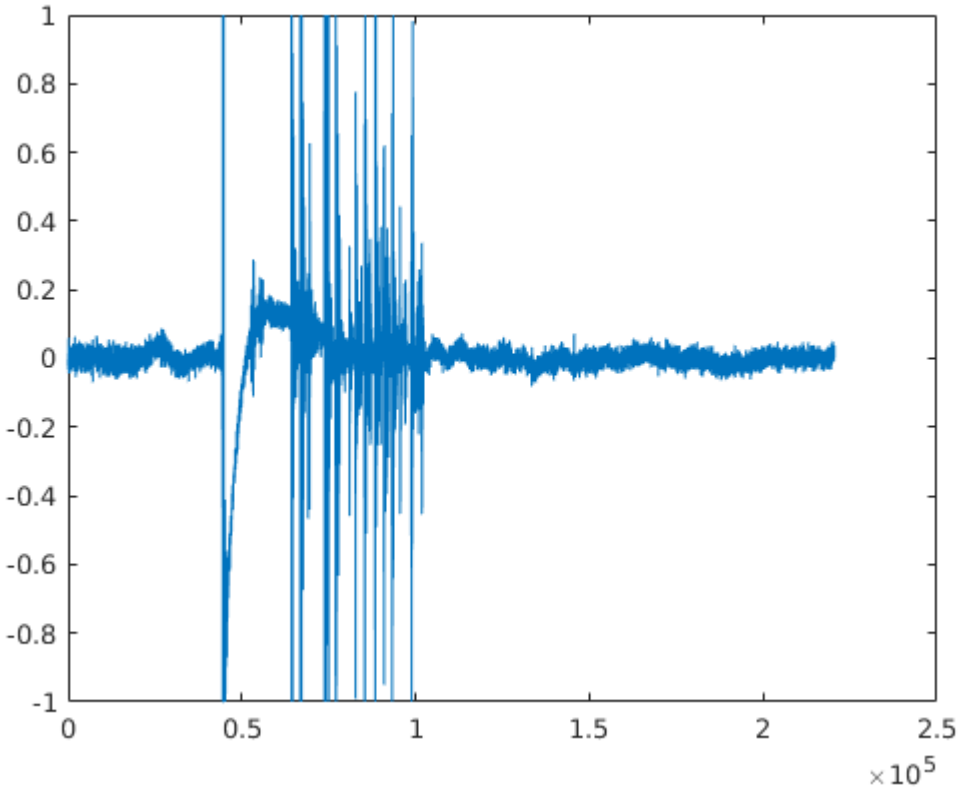
```
In [11]: disp('Start speaking.')
         recordblocking(audio_file, 5);
         disp('End of Recording.');
```

```
Start speaking.
End of Recording.
```

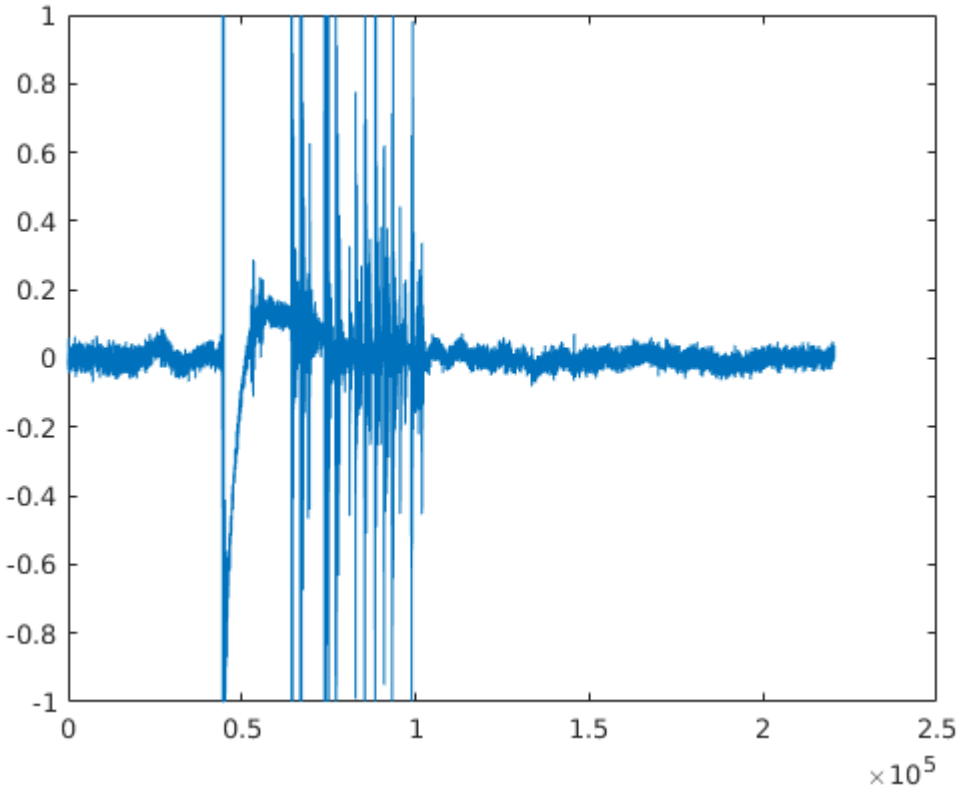
```
In [40]: play(audio_file);
```

```
In [26]: y = getaudiodata(audio_file);
         size_y = size(y,1)
         plot(y);
```

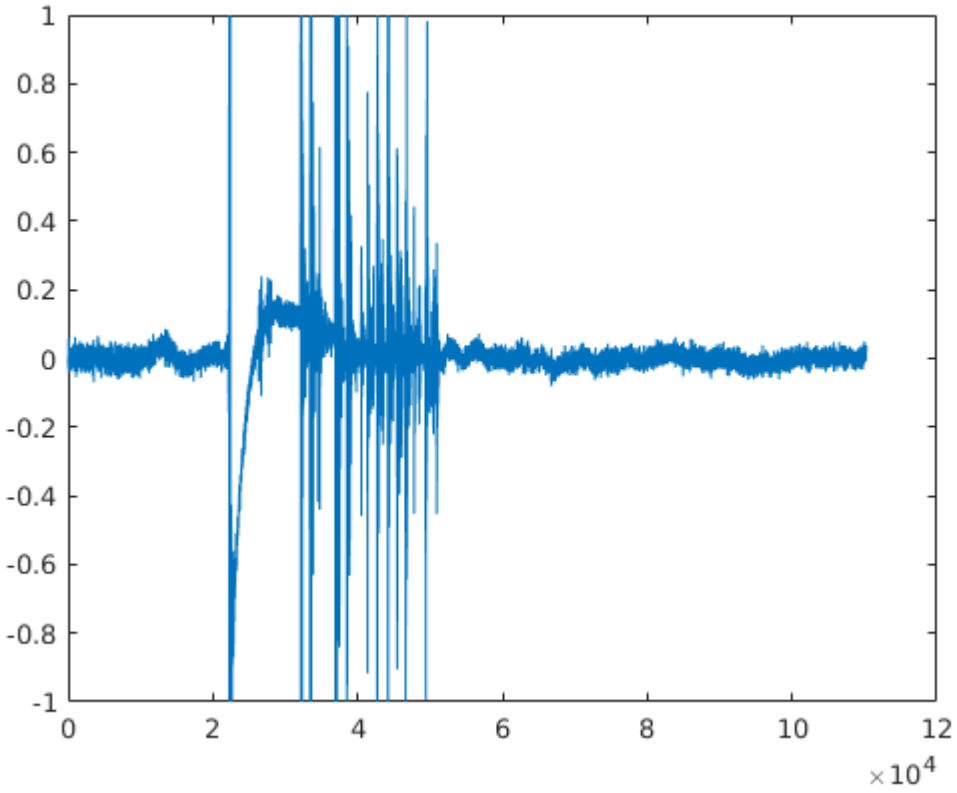
```
size_y =

      220500
```



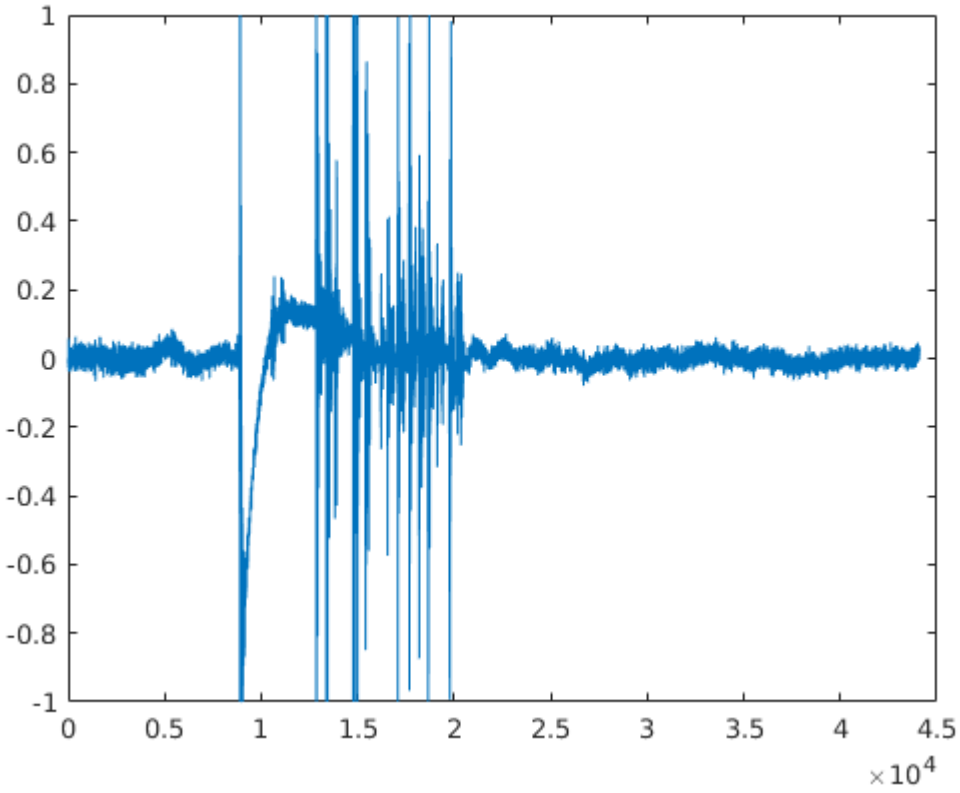

```
In [30]: y24 = y((1:floor(44.1/24):size_y)');
         P24 = audioplayer(y24,24000);
         plot(y24)
         play(P24)
```



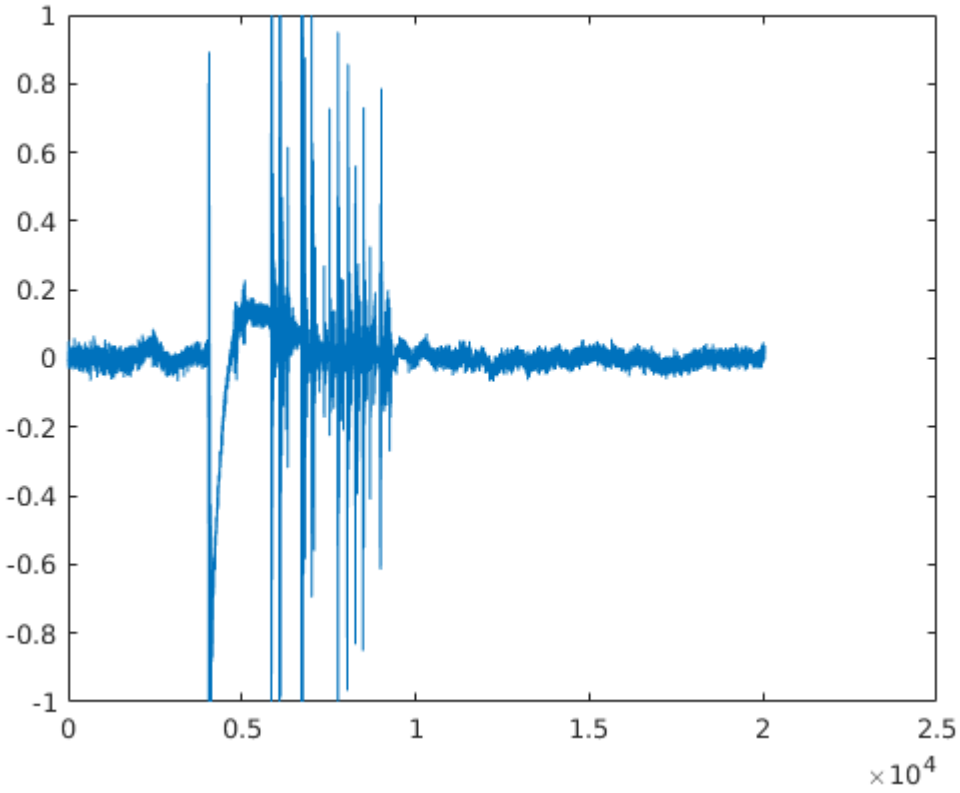
In [31]:
```
y16 = y((1:floor(44.1/16):size_y)');
P16 = audioplayer(y24,16000);
plot(y16)
play(P16)
```



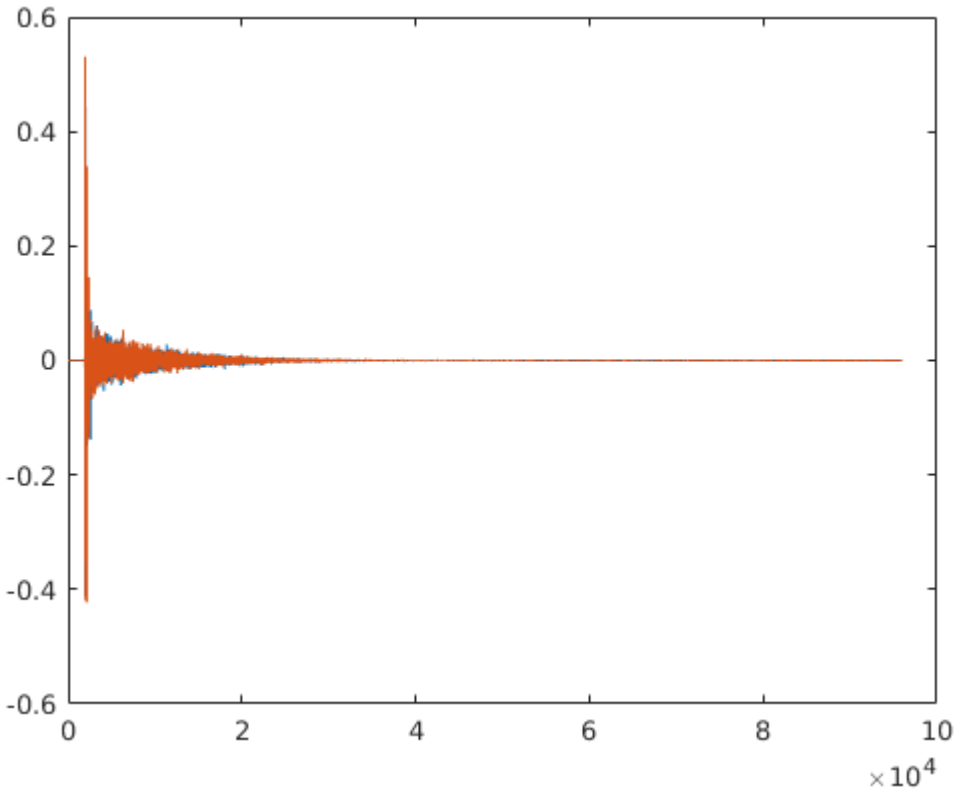

```
In [32]: y8 = y((1:floor(44.1/8):size_y)');
         P8 = audioplayer(y24,8000);
         plot(y8)
         play(P8)
```



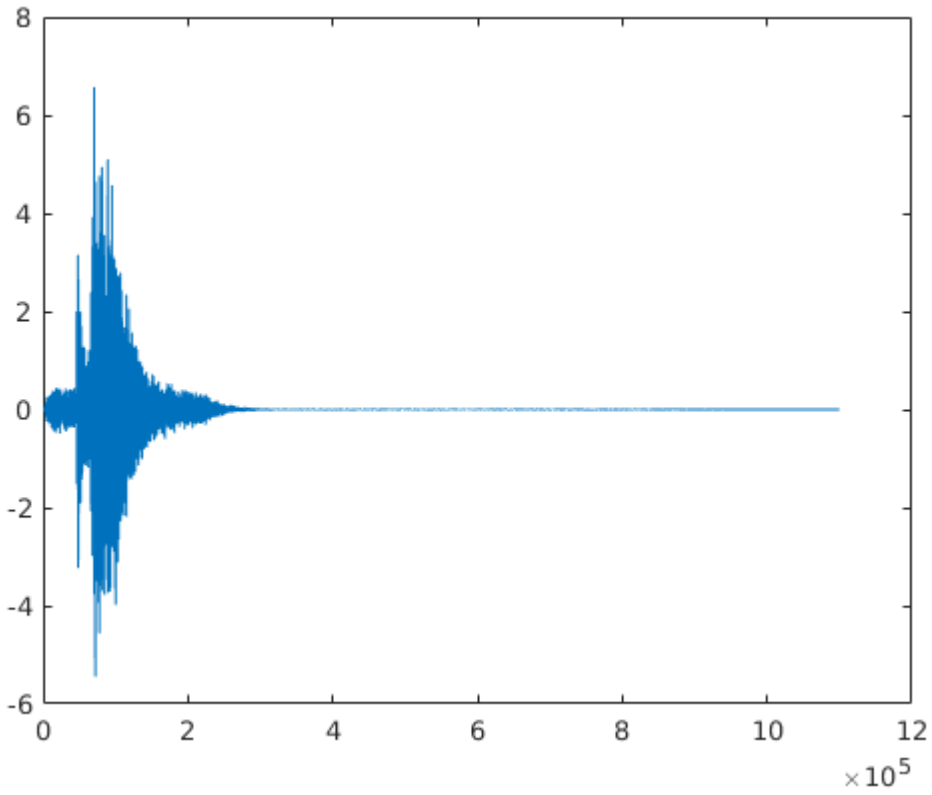
In [33]:
```
y4 = y((1:floor(44.1/4):size_y)');
P4 = audioplayer(y24,4000);
plot(y4)
play(P4)
```



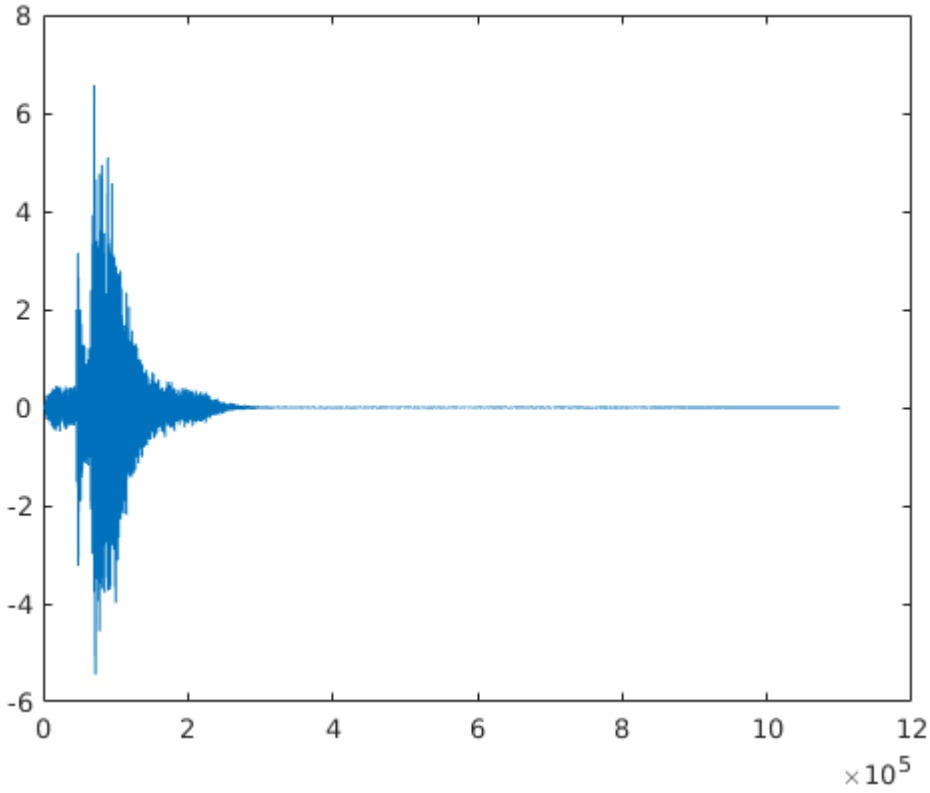
In [34]:
```
church_bullet = audioread('church_bullet.wav');
church = audioplayer(church_bullet,44100);
play(church);
plot(church_bullet);
```



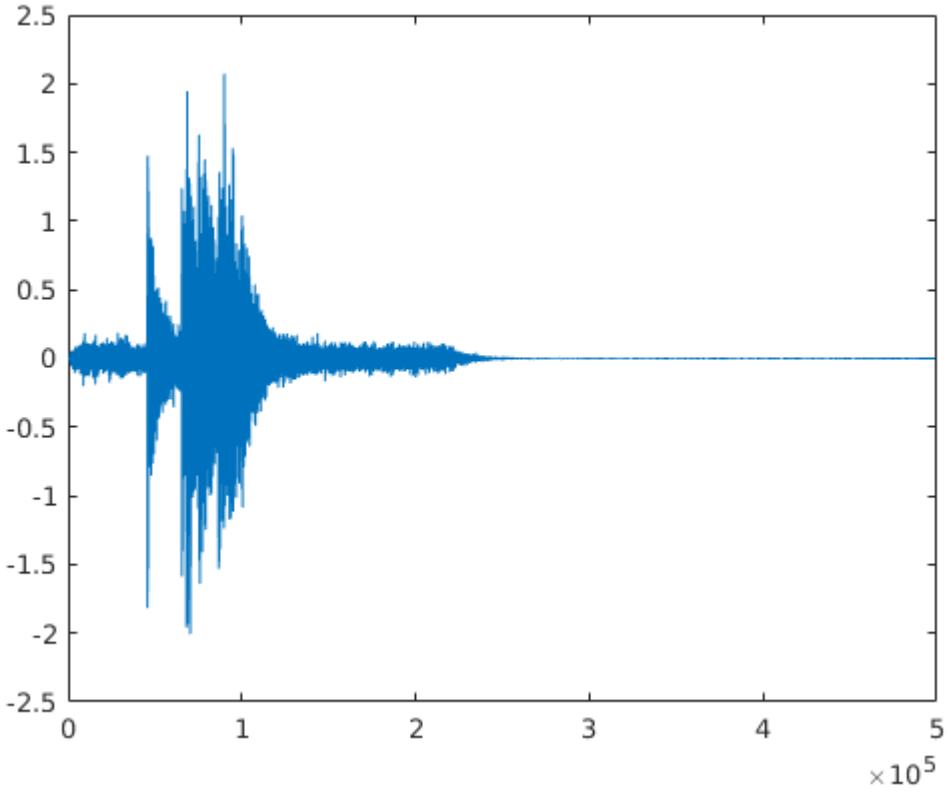
In [38]:
```
church_y = conv(y,church_bullet(:,1));
church_conv_audio = audioplayer(church_y,44100);
plot(church_y)
play(church_conv_audio)
```




```
In [39]: r1_bullet = audioread('./r1_omni.wav');
         r1_y = conv(y,r1_bullet(:,1));
         r1_conv_audio = audioplayer(r1_y,44100);
         plot(r1_y)
         play(r1_conv_audio)
```




```
In [41]: tr_bullet = audioread('./terrys_typing_omni.wav');
         tr_y = conv(y,tr_bullet(:,1));
         tr_conv_audio = audioplayer(tr_y,44100);
         plot(tr_y)
         play(tr_conv_audio)
```



In [ ]:

q5

January 20, 2018

1 2 D Convolution and edge detection

Let assume the below 3x3 matrix

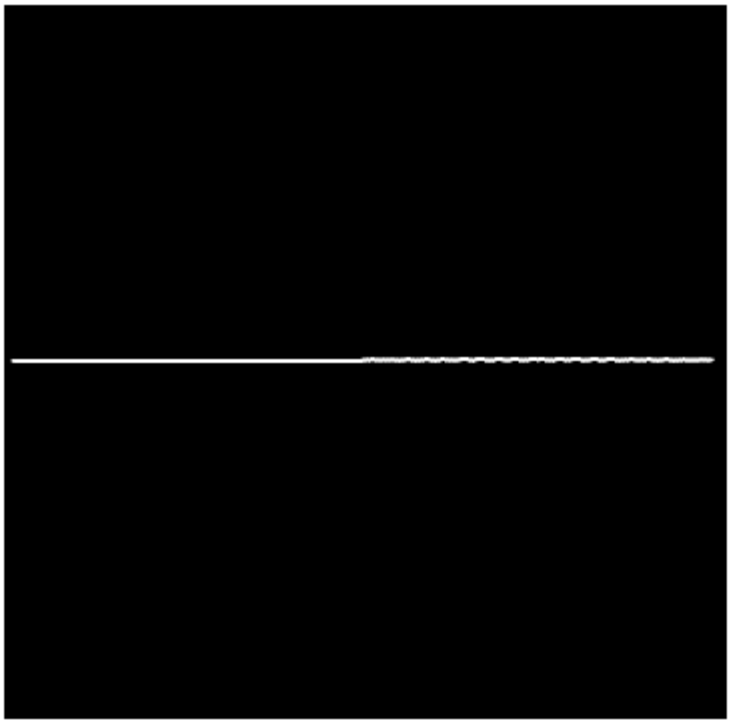
```
In [22]: M = [1,2,1;0,0,0;-1,-2,-1]
```

```
M =

     1     2     1
     0     0     0
    -1    -2    -1
```

We can see after convolution, we get a red line.

```
In [10]: img = imread('./sample_inp.png');
         new_img = myconv(M,img);
```

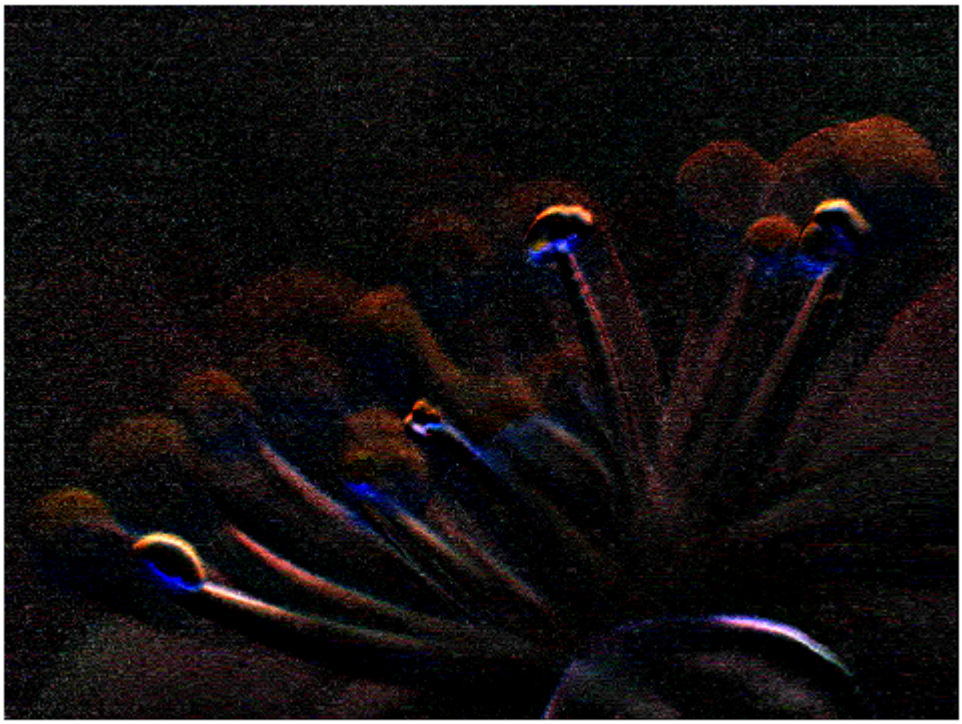
```
In [11]: imshow(new_img);
```



1.0.1 Using the M we first detect many vertical edges

```
In [24]: blur = imread('./blur.jpg');
         ver_blur = myconv(M,blur);
         imshow(ver_blur*5)
```

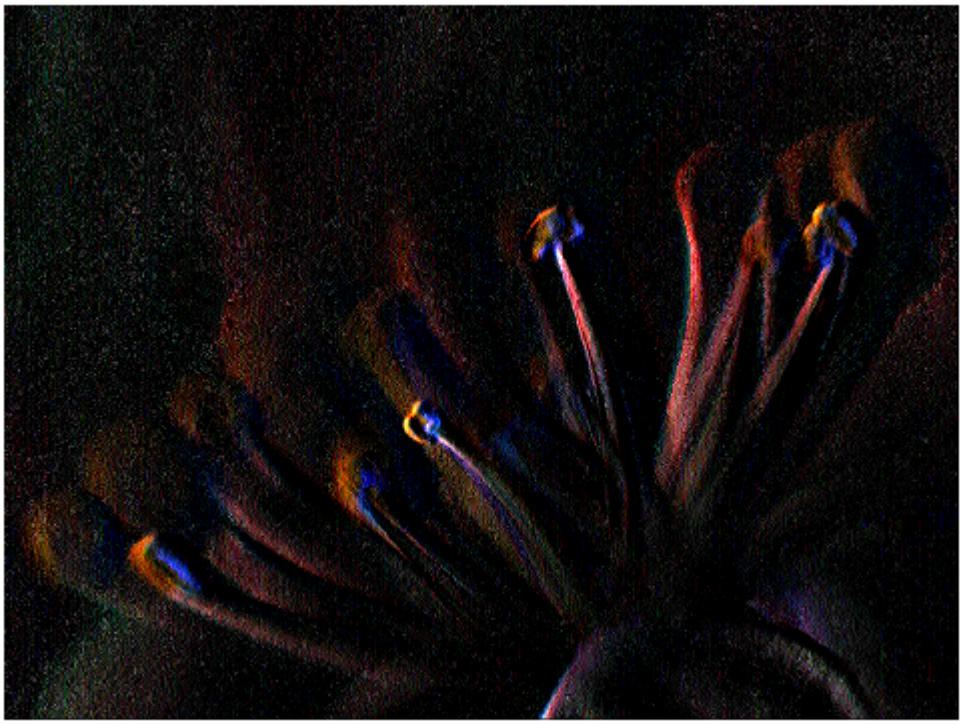
```
Warning: Image is too big to fit on screen; displaying at 67%
> In images.internal.initSize (line 71)
  In imshow (line 328)
```



1.0.2 Using the M' we detect many horizontal edges

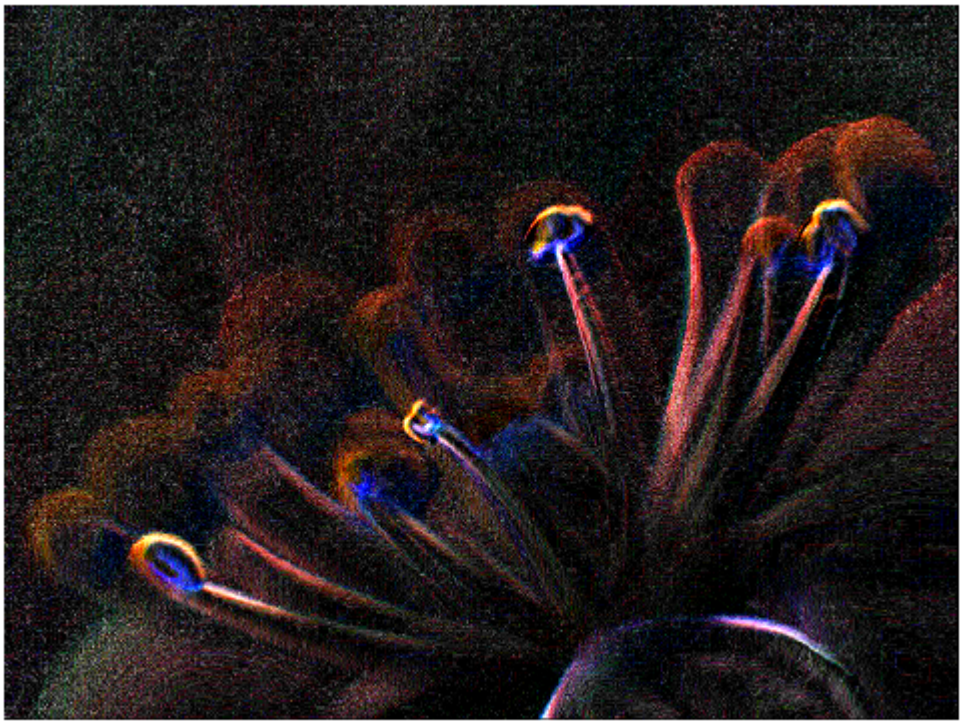
```
In [25]: hor_blur = myconv(M',blur);
         imshow(hor_blur*5)
```

```
Warning: Image is too big to fit on screen; displaying at 67%
> In images.internal.initSize (line 71)
  In imshow (line 328)
```



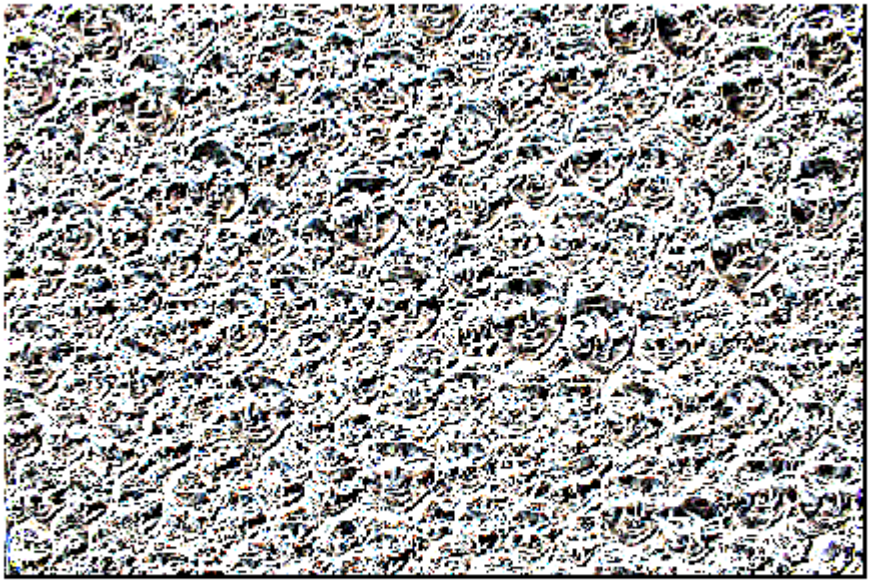
```
In [30]: fin_blur = ver_blur + hor_blur;
         imshow(fin_blur.*5)
```

```
Warning: Image is too big to fit on screen; displaying at 67%
> In images.internal.initSize (line 71)
  In imshow (line 328)
```



Hence we can even do this on other images‘

```
In [33]: img = imread('./Faces.jpg');
         ver_img = myconv(M,img);
         hor_img = myconv(M',img);
         fin_img = ver_img + hor_img;
         imshow(fin_img.*5)
```



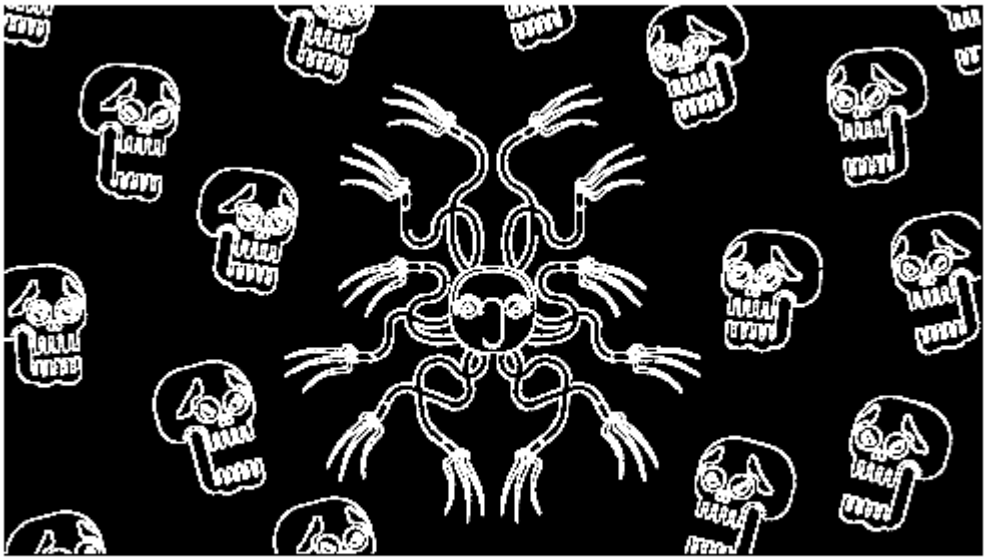
```
In [34]: img = imread('./War_on_drugs.png');
         ver_img = myconv(M,img);
         hor_img = myconv(M',img);
         fin_img = ver_img + hor_img;
         imshow(fin_img.*5)
```

```
Warning: Image is too big to fit on screen; displaying at 67%
> In images.internal.initSize (line 71)
  In imshow (line 328)
```

1.0.3 We get better edge detection after using some kind of blur

```
In [36]: fin_img = conv2([1,1,1;1,1,1;1,1,1]./9,fin_img);
         imshow(fin_img)
```

```
Warning: Image is too big to fit on screen; displaying at 67%
> In images.internal.initSize (line 71)
  In imshow (line 328)
```



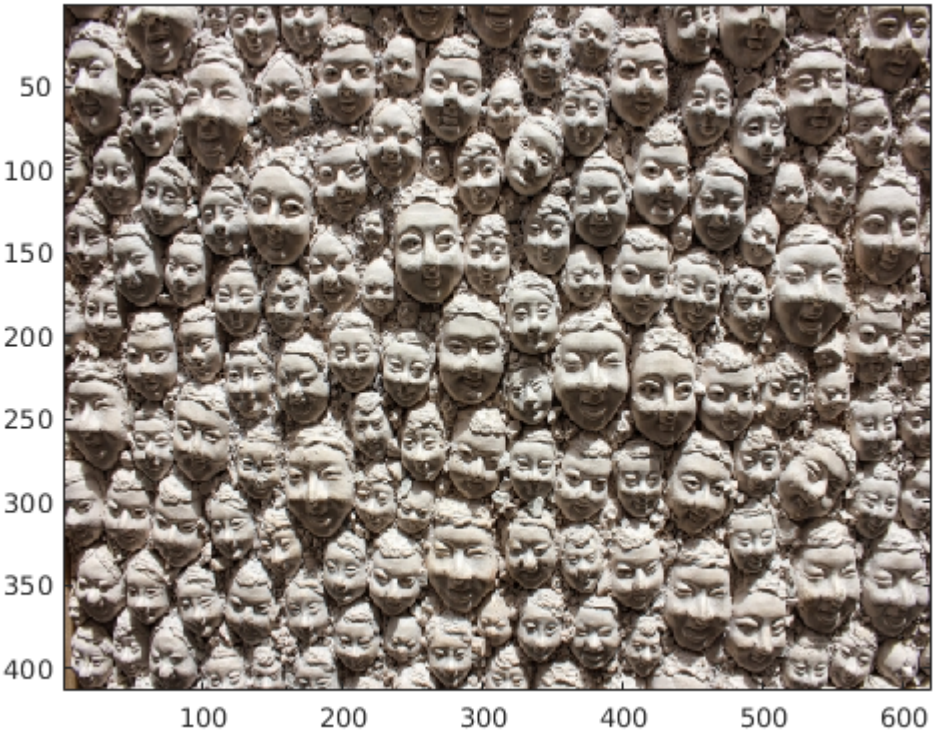
In [ ]:

q6

January 20, 2018

1 Sub Image Detection using Cross Correlation

In [2]:
```
img = imread('./Faces.jpg');
image(img)
img = rgb2gray(img);
```

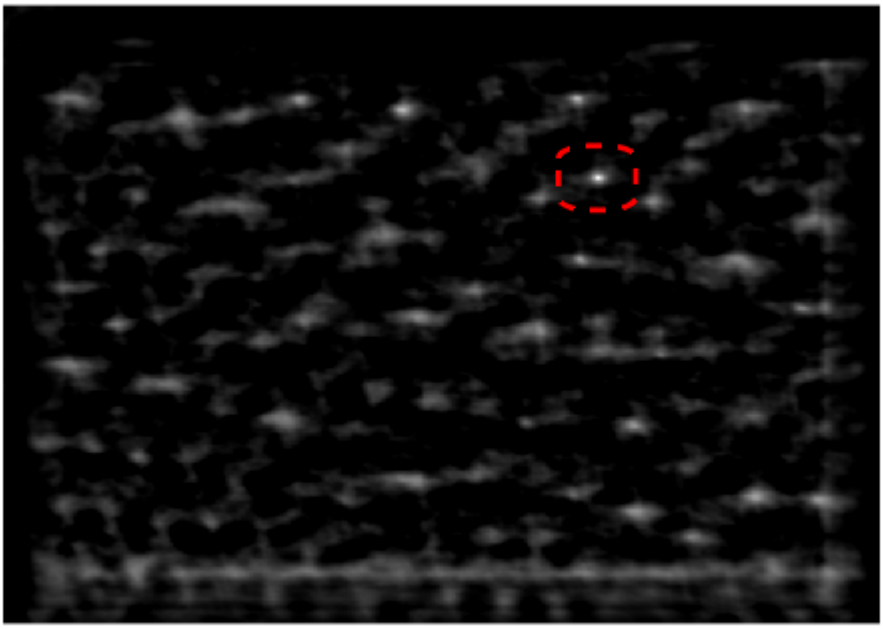


In [3]:
```
sub_img = imread('./F1.jpg');
image(sub_img)
sub_img = rgb2gray(sub_img);
```

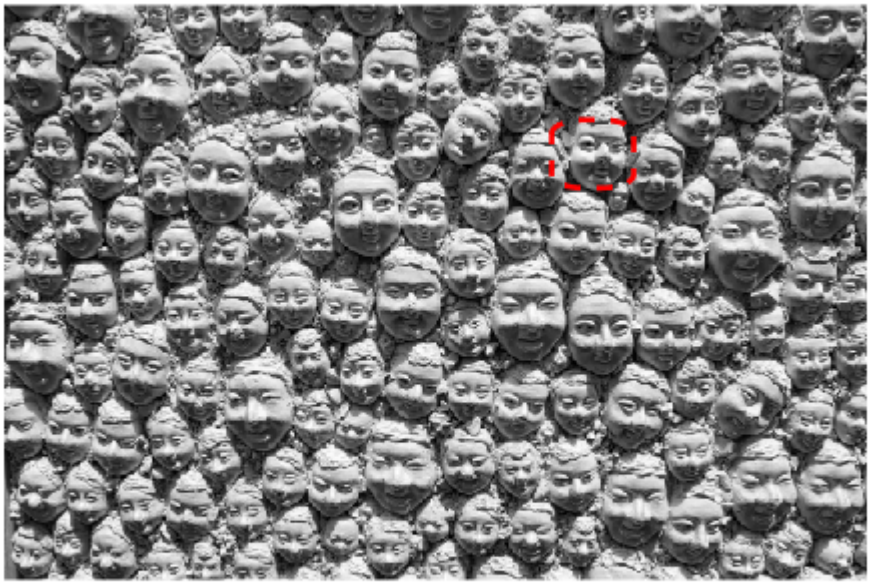

```
In [4]: new_img = normxcorr2(sub_img(:,:,1),img(:,:,1));

In [5]: [val,p] = max(new_img(:));
        x = floor(p/size(new_img,1))
        y = p - x*size(new_img,1)
```

```
x =

   453


y =

   132
```

```
In [6]: imshow(new_img);
        hold on;
        % Then, from the help:
        rectangle('Position',[x - size(sub_img,1)/2,y - size(sub_img,2)/2,size(sub_img,1),size(sub_img,
                  'Curvature',[0.8,0.4],...
                 'LineWidth',2,'LineStyle','--','EdgeColor','r')
```



In [8]:
```
imshow(img);
hold on;
% Then, from the help:
rectangle('Position',[x - size(sub_img,1),y - size(sub_img,2),size(sub_img,1),size(sub_img,2)],
          'Curvature',[0.8,0.4],...
         'LineWidth',2,'LineStyle','--','EdgeColor','r')
```



1.1 Report

- We can see various white spots in the final correlation image
- These dots represent the highly similiar sections with the corresponding sub_image
- Hence we can say we detected the subimage at these locations in the original image

In [ ]:

q7

January 20, 2018

1 Finding the 1-D filter

```
In [3]: inp = [12, 20, 3, 10, 22, 19, 23, 16, 0, 21, 23, 16, 18]
        inp_size = size(inp,2)
```

```
inp =

    12    20     3    10    22    19    23    16     0    21    23    16    18


inp_size =

    13
```

```
In [4]: out = [75, 52, 33, 97, 251, 211, 63, 65]
        out_size = size(out,2)
```

```
out =

    75    52    33    97   251   211    63    65


out_size =

     8
```

1.0.1 Method:

- Using Linear regressio we will try to find the filter
- Hence we are assuming random values in beginning
- We will calculate the pred_out
- We will find the loss and gradient

1.1 Using Normalize equations

```
In [7]: % Assuming stride = 1
        % Assuming padding = 0
        filter_size = (size(inp,2) - size(out,2) + 1);
```

```
In [8]: inp_pred = zeros(filter_size,out_size);
        for i = 1:inp_size - filter_size + 1
            inp_pred(:,i) = inp(i:i+filter_size-1);
        end
        inp_pred
```

```
inp_pred =

    12    20     3    10    22    19    23    16
    20     3    10    22    19    23    16     0
     3    10    22    19    23    16     0    21
    10    22    19    23    16     0    21    23
    22    19    23    16     0    21    23    16
    19    23    16     0    21    23    16    18
```

```
In [9]: filter = out*pinv(inp_pred)
```

```
filter =

    5.0000    4.0000    4.0000   -3.0000   -3.0000    1.0000
```

1.2 Using Gradient Desent

```
In [10]: epoch = 1000;
         lr = 0.001;
         FILTER = rand(1,filter_size).*10
```

```
FILTER =

    8.1472    9.0579    1.2699    9.1338    6.3236    0.9754
```

```
In [18]: for ind = 1:epoch

         out_pred = FILTER*inp_pred;
         loss = sum((out - out_pred).^2)/(2*out_size);
         gradient = (out_pred - out)*inp_pred'/out_size;
         FILTER = FILTER - lr.*gradient;

         end
```

```
In [16]: FILTER = round(FILTER)
         out_pred = FILTER*inp_pred
         out
```

```
FILTER =

     5     4     4    -3    -3     1


out_pred =

    75    52    33    97   251   211    63    65


out =

    75    52    33    97   251   211    63    65
```

```
In [17]: FILTER = flip(FILTER)
```

```
FILTER =

     1    -3    -3     4     4     5
```

```
In [ ]:
```