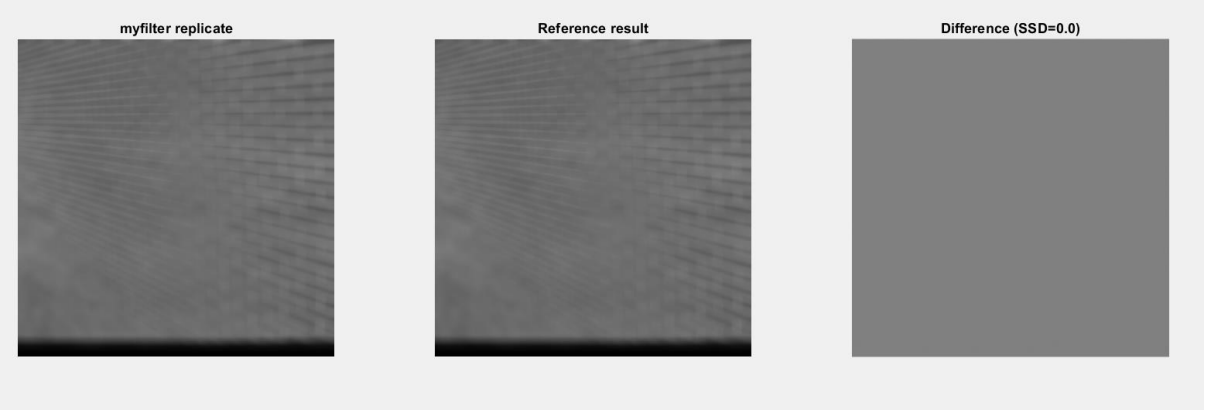
I misread the question and implemented imfilter() on my own ☹.... And since it took me hours to produce the function imfilter(image,kernel,'replicate'), I might as well use it. To test that the function is exactly the same as imfilter. I run this code.

```
image = im2double(imread('brick_wall.tiff'));

kernel = ones(33) / (33*33);

filtered = myfilter_replicate(image, kernel);

reference = imfilter(image, kernel, 'replicate');

difference = 0.5 + 10 * (filtered - reference);

ssd = sum((filtered(:) - reference(:)) .^ 2);

subplot(131); imshow(filtered); title('myfilter replicate');

subplot(132); imshow(reference); title('Reference result');

subplot(133); imshow(difference); title(sprintf('Difference (SSD=%.1f)',ssd));
```

Result:



2.1 Box filter

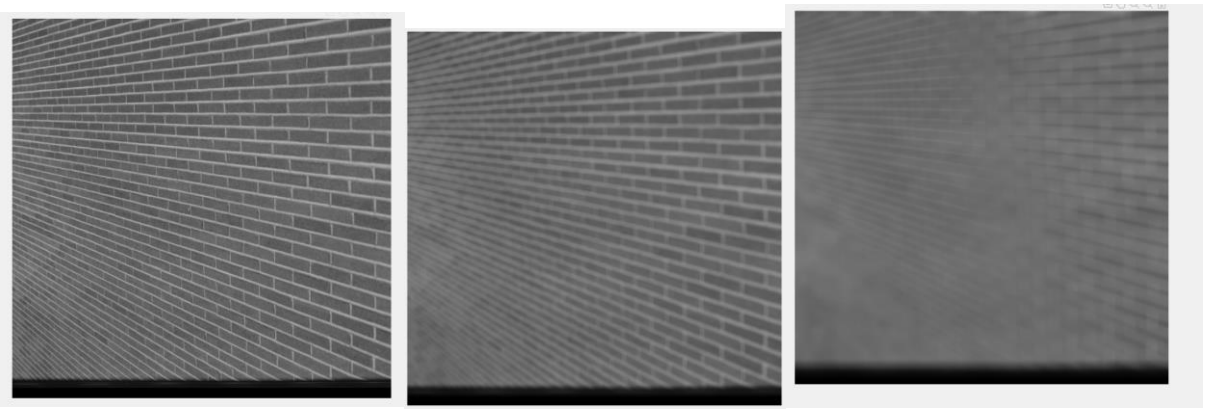
Box filter is implemented with the following function:

```
function mybox = boxfilter(size)

mybox = ones(size);

size = size*size;

mybox = mybox/size;

end
```

To test that it works, I run mybox = boxfilter(3)and the following output is produced.

```
mybox =

    0.1111    0.1111    0.1111
    0.1111    0.1111    0.1111
    0.1111    0.1111    0.1111
```

The increase in the box size would make the resulting image more blur, as all pixels inside the boxes are averaged, the bigger the box filter, the smaller each original pixel value contributes to the final image. You can see the increase in blurness as I increase the size of my boxfilters.



2.1 Gaussian Filter

Gaussian filter is implemented through the following function.

```
function kernel = myGaussian(std)
   %Gaussian 5x5 with sigma 1 using the formula
   kernel = zeros(5);
   sigma = std;
   temp = 0;
   for i= 1: size(kernel,1)
      for j = 1: size(kernel,2)
         a = (i-3)^2+ (j-3)^2;
         kernel (i,j) = exp(-1*(a)/(2*(sigma^2)));
         temp = temp + kernel(i,j);
      end
   end
   %normalise
   kernel = kernel/temp;
end
```

With kernel = myGaussian(1), the following output is produced:

```
0.0030    0.0133    0.0219    0.0133    0.0030
0.0133    0.0596    0.0983    0.0596    0.0133
0.0219    0.0983    0.1621    0.0983    0.0219
0.0133    0.0596    0.0983    0.0596    0.0133
0.0030    0.0133    0.0219    0.0133    0.0030
```

As the standard deviation increases, it continues to blur the intensity of the image, but still retain its high frequency detail such as the edges of the wall. And as sigma is large enough, it will just be a box filter. Below is the gaussian with increasing standard deviation, notice that the picture is blurrer but the edges still retain in the latter image.

2.2 SURF

SURF

The features are extracted with the following codes:

```
image = imread('GOPR1515 03850.jpg');
grayImage = rgb2gray(image);


points = detectSURFFeatures(grayImage)


image2 = imread('GOPR1515 03852.jpg');
grayImage2= rgb2gray(image2);
points2 = detectSURFFeatures(grayImage2)


[feature1, validPoints1] = extractFeatures(grayImage, points);
[feature2, validPoints2] = extractFeatures(grayImage2, points2);
```

I have calculate the distance of each features using the following codes, and sort it in ascending to find the pixel with the nearest distance:

```
threshold = 0.5;
for i = 1:size(feature1,1)
   euclid = zeros(size(feature2,1));
   for j=1:size(feature2,1)
        euclid(j) = sqrt(sum((feature1(i,:)-feature2(j,:)).^2));
   end


   [vals,indx] = sort(euclid);
   if (vals(1) < threshold * vals(2))
     match(i) = indx(1);
   else
     match(i) = 0;
   end
end
```

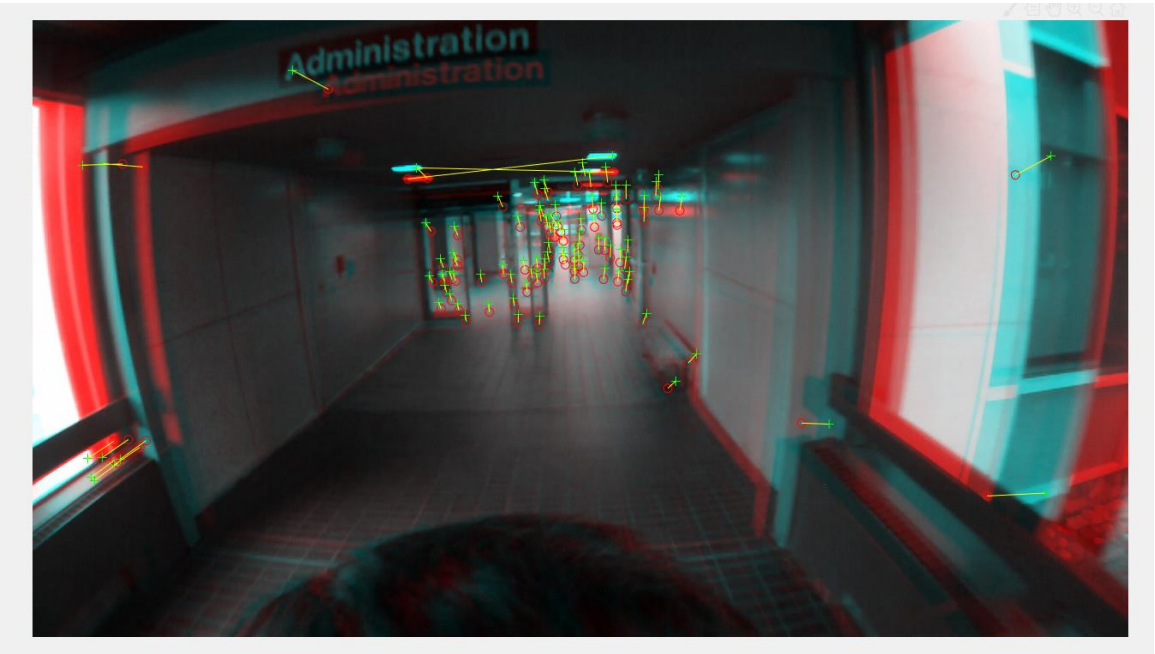
In order to get the same type of data produced by the matlab matchFeatures() I run the following:

```
for i = 1:size(match,2)
   if match(1,i)== 0
      continue
   else
      myindexpairs(i,1)=i
      myindexpairs(i,2)=match(1,i)
   end
end
myindexpairs( all(~myindexpairs,2), : ) = []; %Remove the row
```

To show the images:

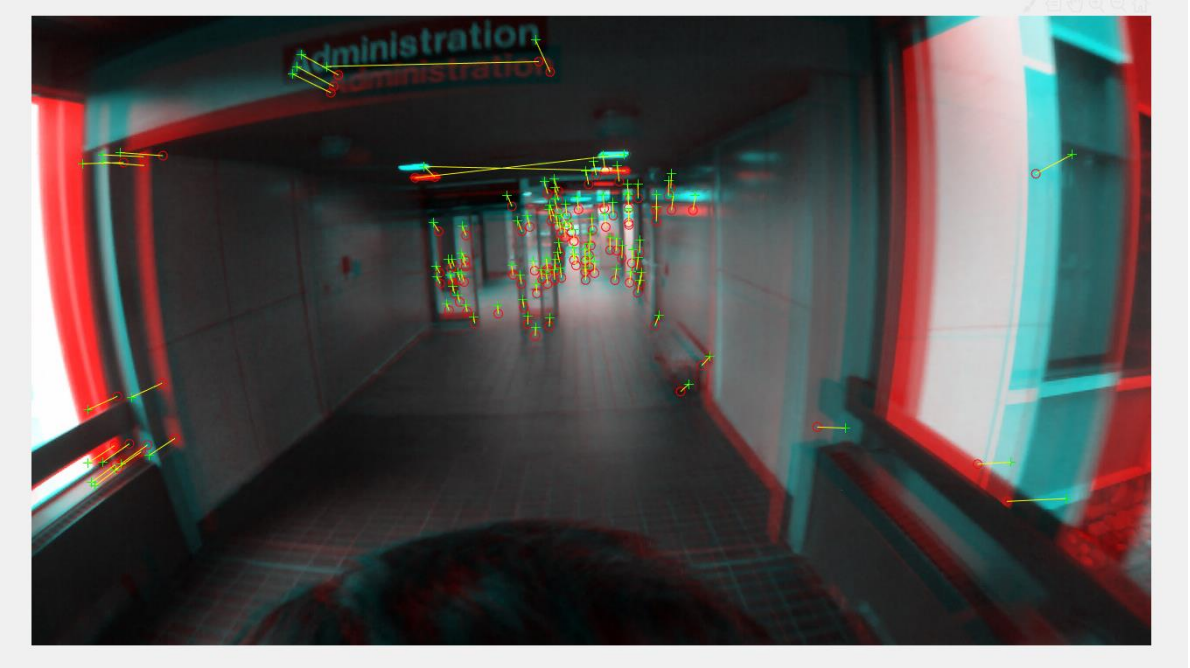
```
matchedPoints1 = validPoints1(myindexpairs(:,1),:);
matchedPoints2 = validPoints2(myindexpairs(:,2),:);
figure; showMatchedFeatures(image,image2,matchedPoints1,matchedPoints2);
```

Result of my own implementation with threshold = 0.5:



```
%Matlab
indexPairs = matchFeatures(feature1,feature2);
matchedPoints1 = validPoints1(indexPairs(:,1),:);
matchedPoints2 = validPoints2(indexPairs(:,2),:);
figure; showMatchedFeatures(image,image2,matchedPoints1,matchedPoints2);
```

Result of matlab inbuilt function matchFeatures()



The first result we produced is pretty decent, it matches most of the features shown using the matlab function, to improve the result, I guess I can improve the result by limiting the amount of matches each feature point can have so that they don't match with more than one point. E.g. the ceiling lights match with themselves and to each other.

2.3 Fast Fourier Transform

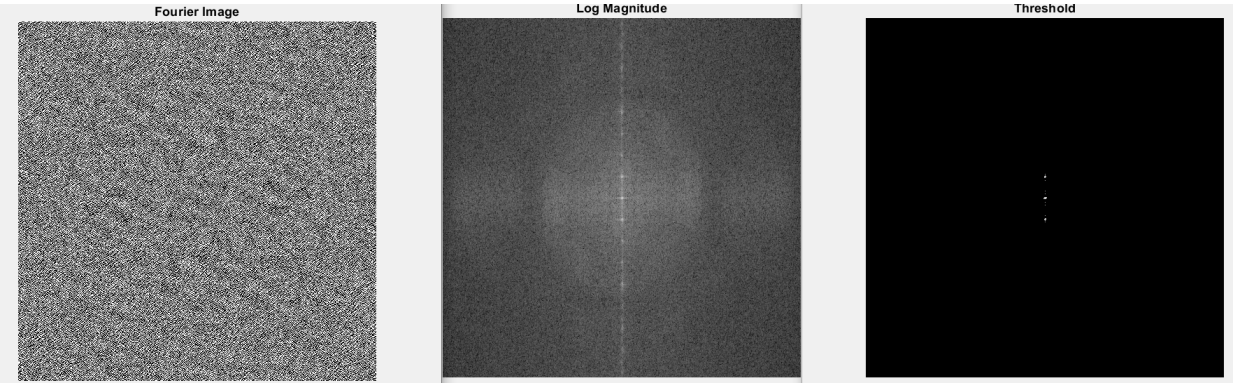
The FFT2() is implemented with the following codes:

```
image = double(imread('son3.gif'));
figure,imshow(image)
title('original')
[X, Y] = size(image);
disp(X)
disp(Y)
imgX = zeros(X, X);
imgY = zeros(Y, Y);
for u = 0 : (X - 1)
   for x = 0 : (X - 1)
      imgX(u+1, x+1) = exp(-2 * pi * 1i / X * x * u);
   end
end
for v = 0 : (Y - 1)
   for y = 0 : (Y - 1)
      imgY(y+1, v+1) = exp(-2 * pi * 1i / Y * y * v);
   end
end
F = imgX * double(image) * imgY;
figure,imshow(F)
title('Fourier Image')
F = fftshift(F);
F = abs(F); % Get the magnitude
F = log(F + 1);
figure, imshow(F*255,[])
disp(F)
title('Log Magnitude')
threshold = 8.0
```

F(abs(F)<threshold) = 0;

figure,imshow(F*255,[])

title('Threshold')



Rotation is done through:

%image = imrotate(image,30)

%image = imrotate(image,60)

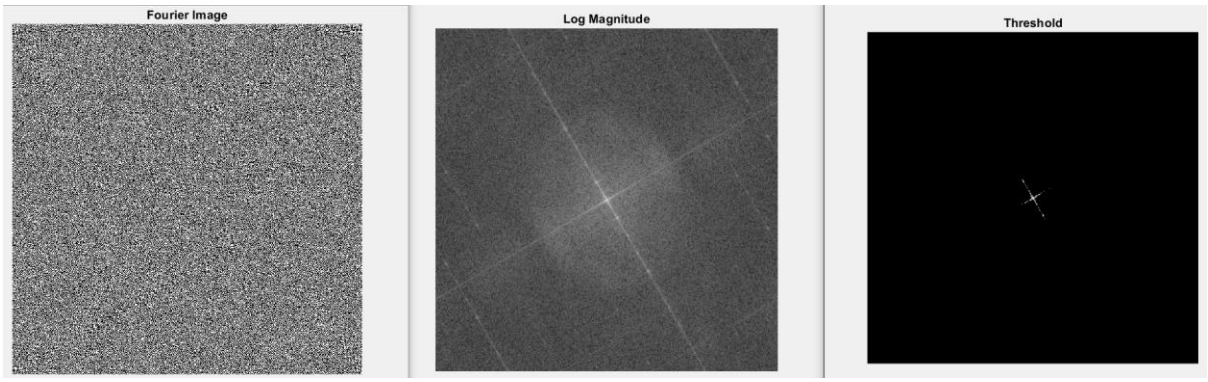
%image = imrotate(image,90)

%image = imrotate(image,120)

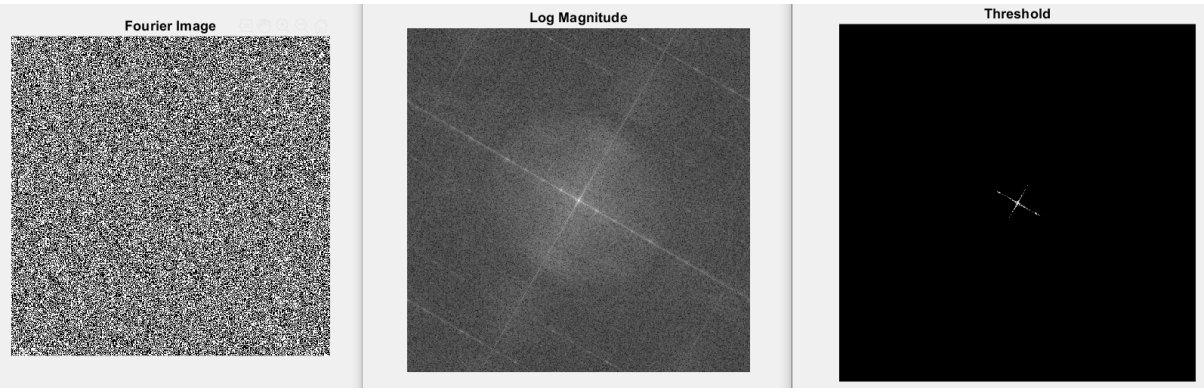
Discussion: We could see the orientation of the text from the lines of the main peaks in the fourier domain, the line is rotated according to the rotation of the input image. Fourier transform represents the image as series of sin/cos waves, and since sin (0) = 0 and cos (0) = 1, the strength of the frequency responses for the sin and cos wave would indicate the dominating directions of the image. For example, if an image were aligned properly, we would see a strong frequency response from either the sin or cos wave, which in turn tells us the dominating directions of the fourier image, and hence allow us to indicate the orientation of the text.

Results:

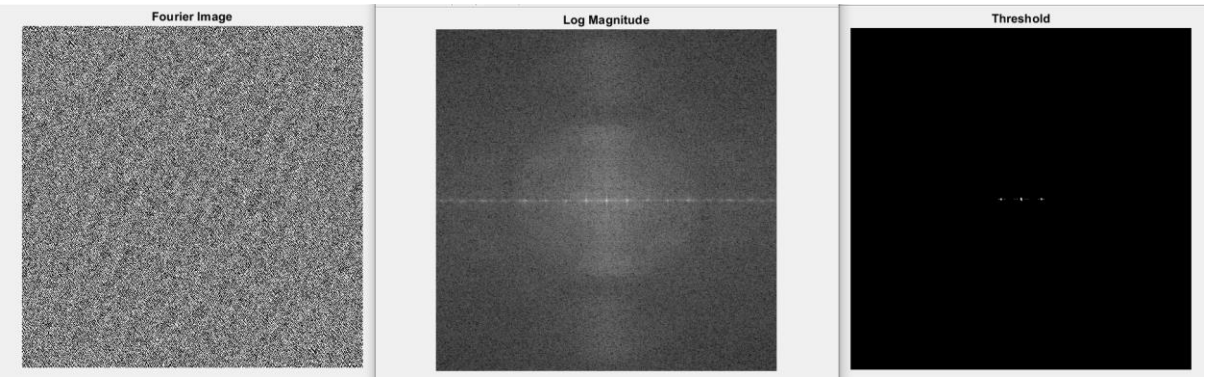
30 degree anti clockwise



60 degree anti clockwise



90 degree anti clockwise



120 degree anticlockwise