Fast Object Tracking

Dr. Daniel B. Rowe
Professor of Computational Statistics
Department of Mathematical and Statistical Sciences
Marquette University

Outline

Introduction

We previously learned that instead of moving a kernel in the time/space domain, we can forward discrete Fourier transform, multiply, and inverse discrete Fourier transform back.

It turns out that convolution via the discrete Fourier transform in frequency space is MUCH faster than convolution in image space.

We will use this property for faster convolution computation and for template correlation matching in particular.

We can calculate each of the statistic images for template matching much faster.

Introduction

Imagine I set up a camera to monitor my neighbor driving his sports car.

I watched him drive by once and took a picture for a template.

Introduction

I set it up so that my camera's live feed is sent to my computer.

I want to know every time that he leaves and arrives home.

I will use my template to track his movements.

I also did the same for all of my other neighbors.

Introduction

Here is one instance of his coming home.

General Properties:
Name: 'shelby.mp4'
Path: 'C:\MATH4931\'
Duration: 4.6290
CurrentTime: 4.6290
NumFrames: 274

Video Properties:
Width: 1920
Height: 1080
FrameRate: 59.1877
BitsPerPixel: 24
VideoFormat: 'RGB24'

Play Video Shelby500.avi

The Convolution Theorem

We can use convolution in image space to compute a local average image.

$\overline{x}_{25}$ Smoothed

=

convolution

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

/25

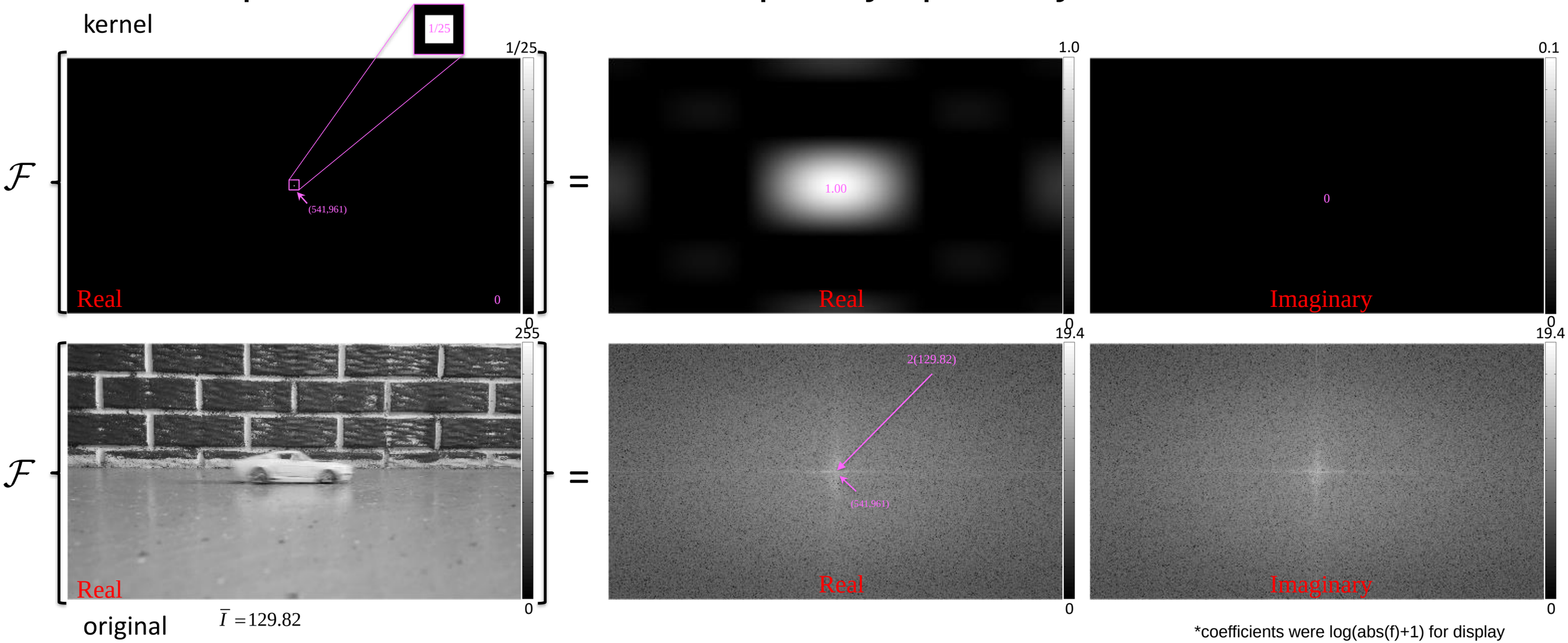
Kernel

$*$

x_i Original

The Convolution Theorem

Or we can perform convolution in frequency space by

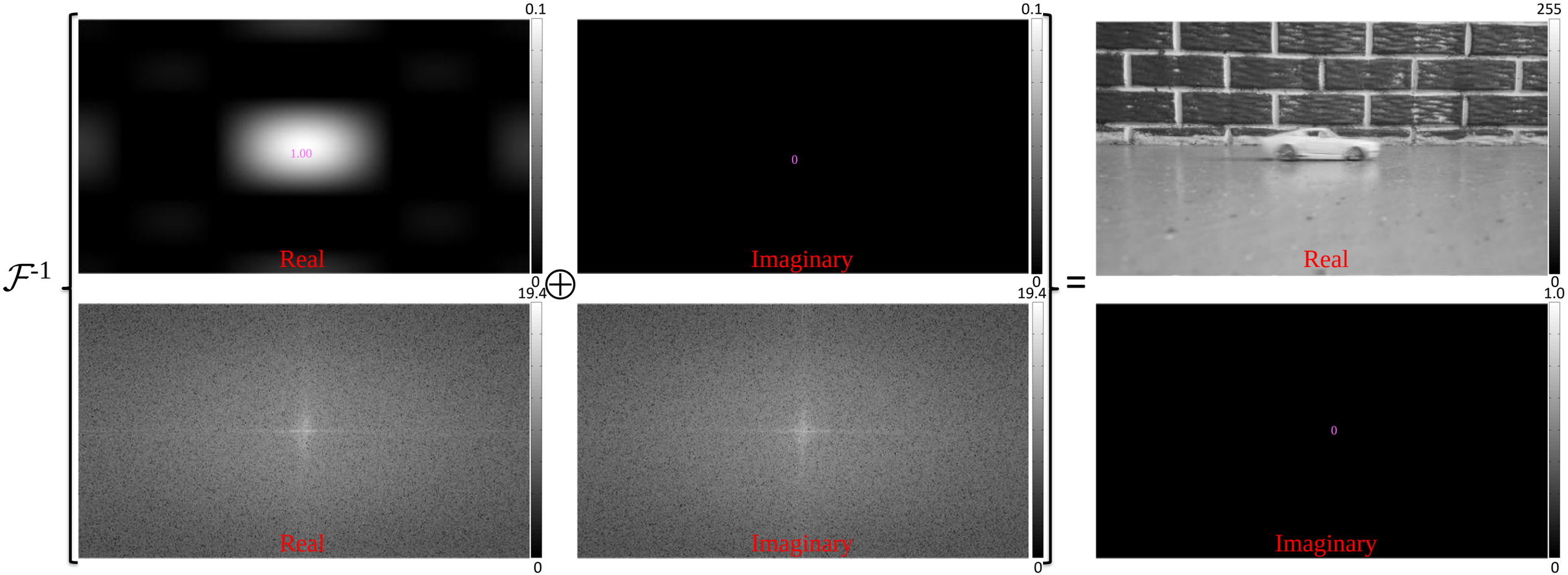


*coefficients were log(abs(f)+1) for display

The Convolution Theorem

⊕ = direct product, element -wise multiplication

… multiplying the two forward discrete Fourier transform together and IDFT.



*coefficients were log(abs(f)+1) for display

The Convolution Theorem

```
limMax=255;
% video file name
filename='shelby.mp4';
% get video information
vidObj = VideoReader(filename);
nt = vidObj.NumFrames;
nx = vidObj.Width;
ny = vidObj.Height;
fr = vidObj.FrameRate;
% convert video to grayscale and equalize
videoRGB = read(vidObj,[1 nt]);
I=zeros([ny,nx,nt]);
for t=1:nt
    I(:,:,t)=(rgb2gray(videoRGB(:,:,:,t)));
end
clear videoRGB
figure; % display video frames
for t=1:nt
    imagesc(I(:,:,t))
    colormap(gray), axis image, axis off
    title(['Frame ',num2str(t)])
    pause(.1)nd
```

```
t=140; Y=I(:,:,t);
figure; % select one frame for convolution
imagesc(Y,[0,limMax])
colormap(gray), axis image, axis off
if (printfigs==1)
    print(gcf,'-dtiffn','-r100',['I',num2str(t)])
end
kernel=ones(5,5)/25; % form kernel for smoothing
[ky,kx]=size(kernel);
% appends border pixels for wrap-around
YW=[Y(ny-(ky-1)/2+1:ny,nx-(kx-1)/2+1:nx),Y(ny-(ky-
1)/2+1:ny,1:nx),Y(ny-(ky-1)/2+1:ny,1:(kx-1)/2);...
    Y(1:ny             ,nx-(kx-1)/2+1:nx),Y(1:ny
,1:nx),Y(1:ny             ,1:(kx-1)/2);...
    Y(1:(ky-1)/2       ,nx-(kx-1)/2+1:nx),Y(1:(ky-1)/2
,1:nx),Y(1:(ky-1)/2       ,1:(kx-1)/2)];
Ysm=zeros(ny,nx); % perform convolution
for j=1:ny
    for i=1:nx
       Ysm(j,i)=sum(sum(kernel.*YW(j:j+ky-1,i:i+kx-1)));
    end
end
figure;,imagesc(Ysm,[0,limMax])
colormap(gray), axis image, axis off
```

The Convolution Theorem

```
kernelfill=zeros(ny,nx);
if (mod(ky,2)==1)
 kernelfill(ny/2-(ky-1)/2+1:ny/2+(ky-1)/2+1,...
     nx/2-(kx-1)/2+1:nx/2+(kx-1)/2+1)=kernel;
elseif (mod(ky,2)==0)
    kernelfill(ny/2-ky/2+1:ny/2+ky/2,nx/2-
kx/2+1:nx/2+kx/2)=kernel;
end
figure;
imagesc(kernelfill,[0,1/25])
colormap(gray), axis image, axis off
figure;
imagesc(kernelfill(ny/2-3:ny/2+5,...
    nx/2-3:nx/2+5),[0,1/25])
colormap(gray), axis image, axis off

ftkern=fftshift(fft2(fftshift(kernelfill)));
maxftKern=max(max(ftkern));
figure;
imagesc(real(ftkern),[0,1])
axis image, colormap(gray), axis off
figure;
imagesc(imag(ftkern),[0,.1])
axis image, colormap(gray), axis off
```

```
ftY=fftshift(fft2(fftshift(Y)));
maxftY=max(max(real(ftY)));
figure;
imagesc(2*real(ftY)/(ny*nx),[0,log(abs(maxftY)+1)/50])
axis image, colormap(gray), axis off
figure;
imagesc(2*imag(ftY)/(ny*nx),[0,log(abs(maxftY)+1)/50])
axis image, colormap(gray), axis off

YsmFT=fftshift(ifft2(fftshift(ftY.*ftkern)));
figure;
imagesc(real(YsmFT),[0,limMax])
colormap(gray), axis image, axis off
figure;
imagesc(imag(YsmFT),[0,1])
colormap(gray), axis image, axis off
```

Template Matching via DFT

$$s^2 = \frac{\sum x_i^2 - \left(\sum x_i\right)^2 / n}{n-1}$$

We are going to use the DFT property to track cars with template matching.

For our object template we can pre-compute

$\sum o_i$ $\sum o_i^2$

pre-compute once

compute each frame for all neighborhoods

Then using the DFT compute

$\sum p_i$ $\sum p_i^2$ $\sum o_i p_i$

$$r = \frac{\sum p_i o_i - \frac{1}{n}\left(\sum p_i\right)\left(\sum o_i\right)}{\sqrt{\sum p_i^2 - \frac{1}{n}\left(\sum p_i\right)^2}\sqrt{\sum o_i^2 - \frac{1}{n}\left(\sum o_i\right)^2}}$$

Template Matching via DFT

$$s^2 = \frac{\sum x_i^2 - \left(\sum x_i\right)^2 / n}{n-1}$$

For our object template we can pre-compute template sums

152×532

$$\sum o_i = 10358226 \qquad \sum o_i^2 = 1929989924 \qquad \text{var}(o_i) = 7459.024811211711$$

pre-compute once

compute each frame for all neighborhoods

Then what we need are the sums for each pixel.

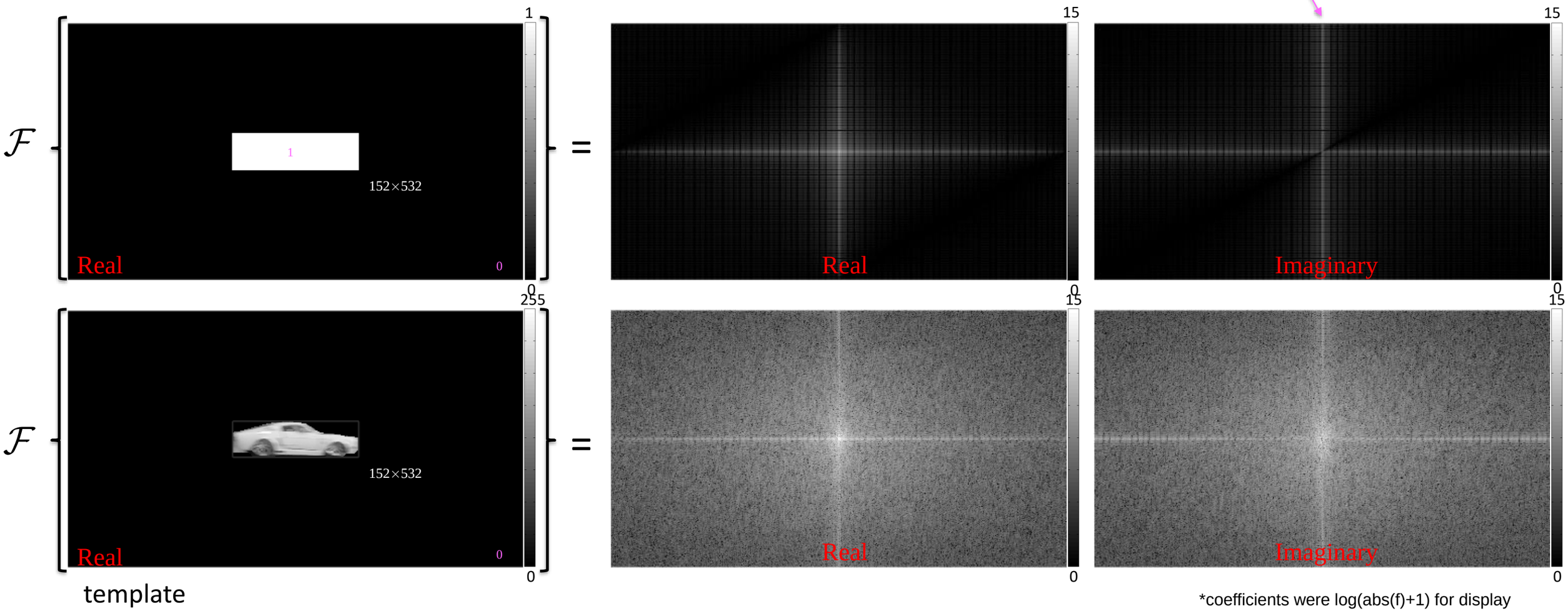
$$\sum p_i \qquad \sum p_i^2 \qquad \sum o_i p_i$$

Template Matching via DFT

imag part because even sized template

We need a centered kernel of ones and template with their DFT for sums.

kernel



template

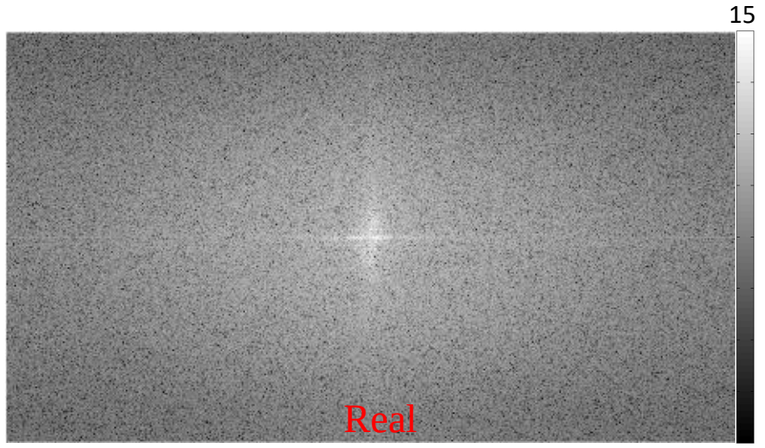
*coefficients were log(abs(f)+1) for display

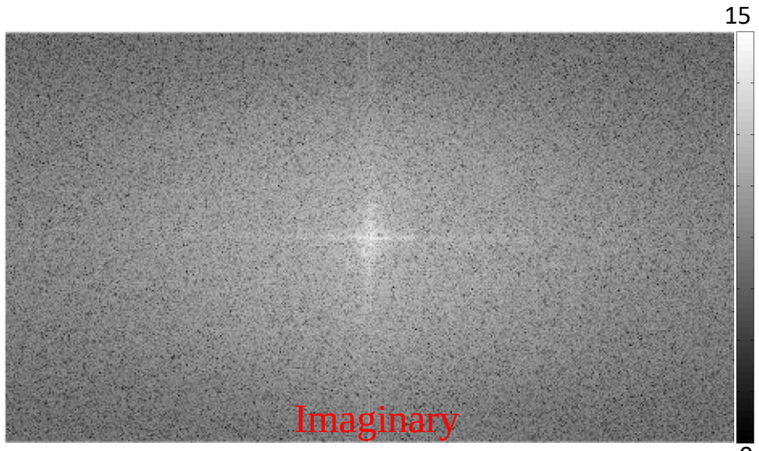
Template Matching via DFT

We need to forward DFT the image and image square.

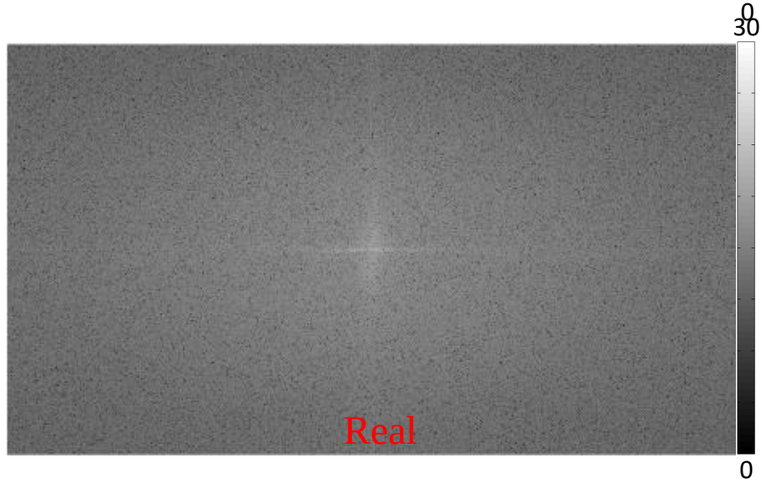
image

$\mathcal{F}$ =





$\mathcal{F}$ =



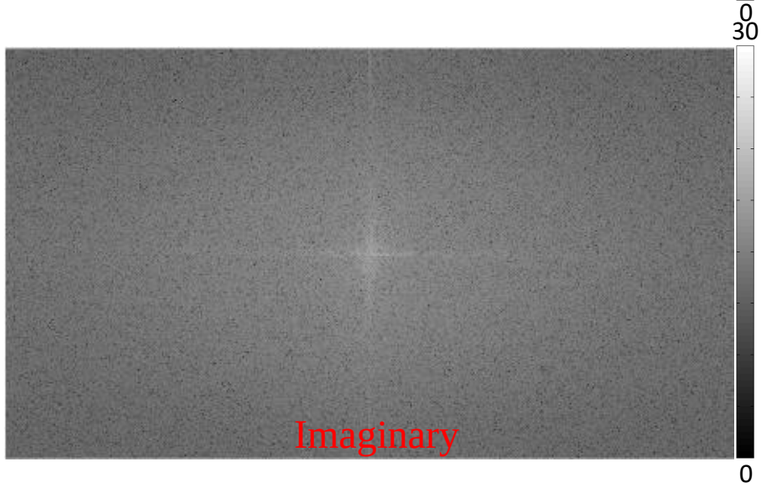
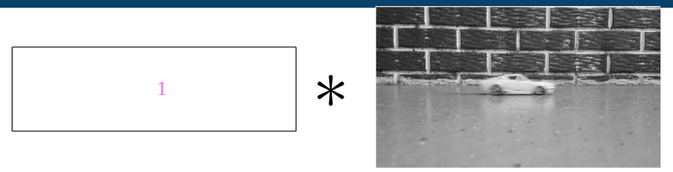


Image-square

*coefficients were log(abs(f)+1) for display

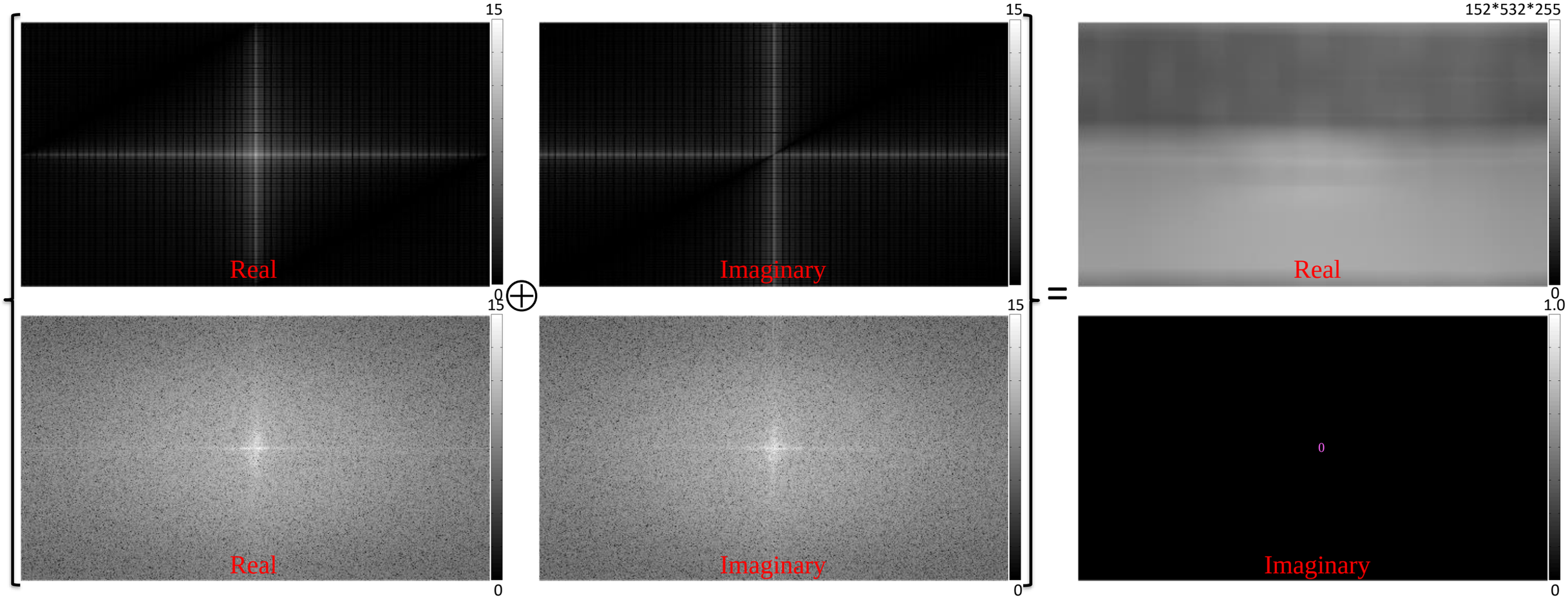
Template Matching via DFT



$\oplus$ = direct product, element -wise multiplication

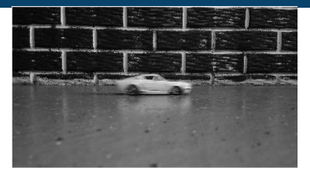
… multiply the appropriate forward DFTs together and inverse DFT. $\sum p_i$

$\mathcal{F}^{-1}$



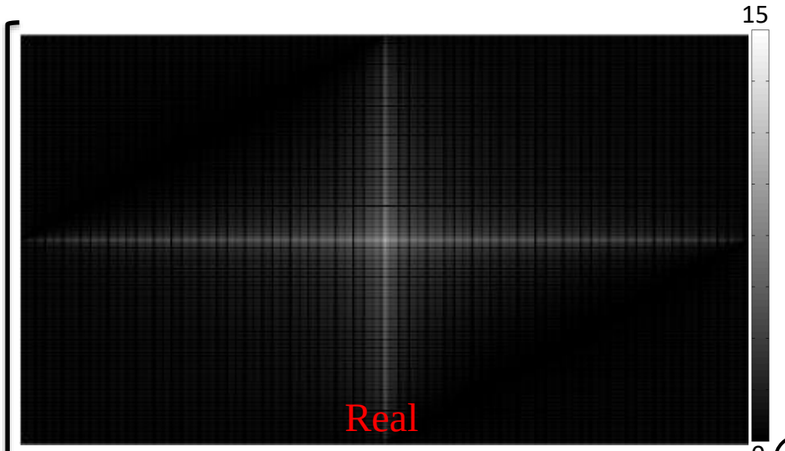
*coefficients were log(abs(f)+1) for display

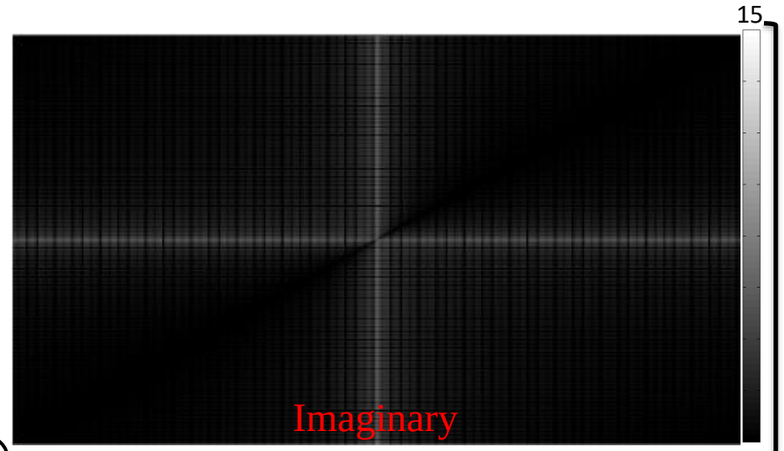
Template Matching via DFT

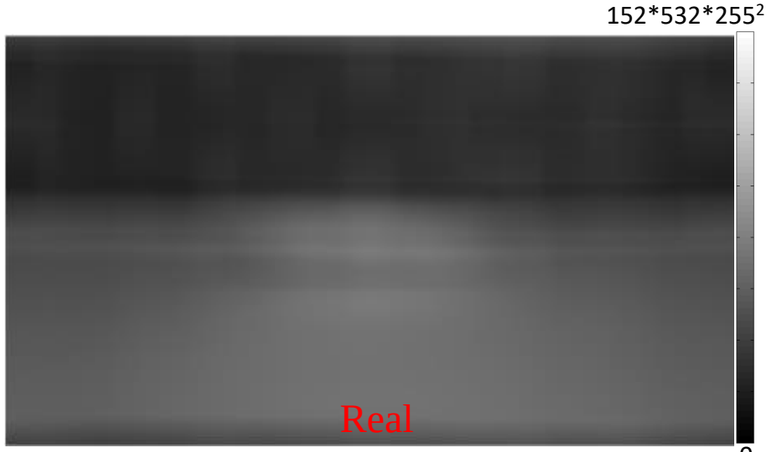
$*$ 

$\oplus$ = direct product, element -wise multiplication

… multiply the appropriate forward DFTs together and inverse DFT. $\sum p_i^2$

$\mathcal{F}^{-1}$ { 

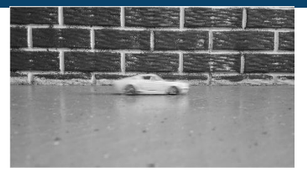
$\oplus$ 

} = 

*coefficients were log(abs(f)+1) for display

Template Matching via DFT

$*$

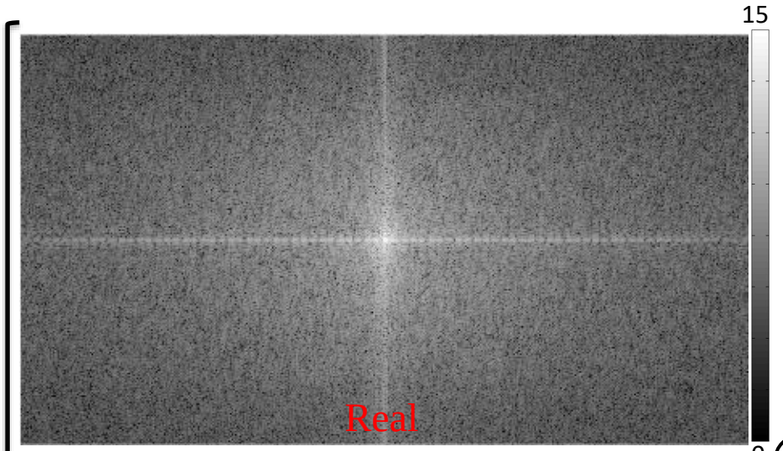


$\oplus$ = direct product, element -wise multiplication

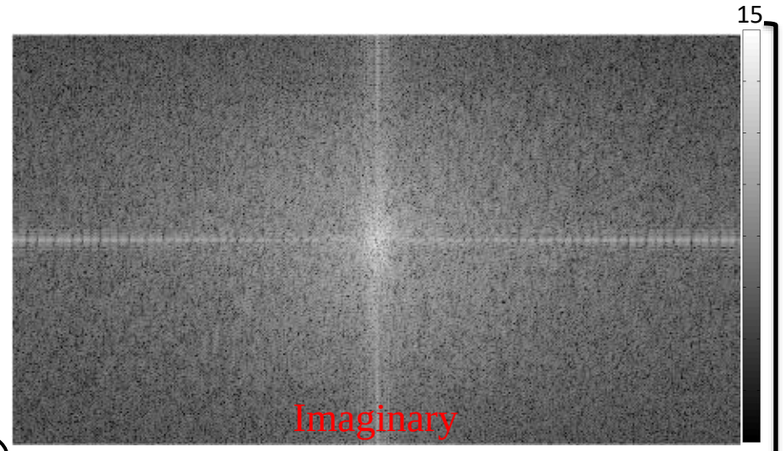
… multiply the appropriate forward DFTs together and inverse DFT.

$$\sum o_i p_i$$

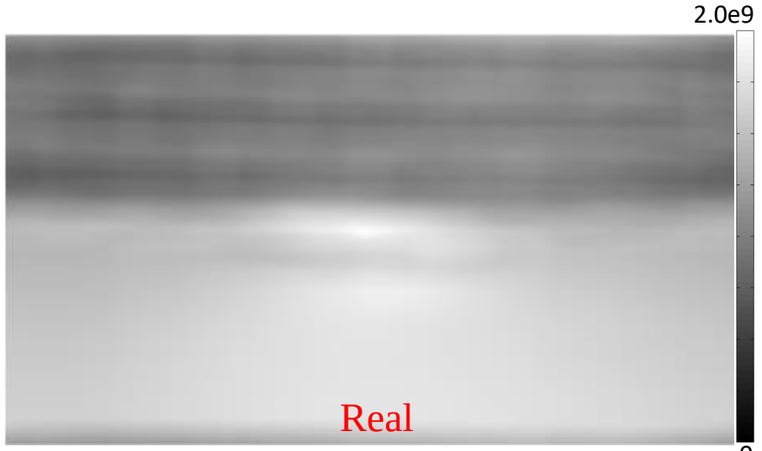
$\mathcal{F}^{-1}$

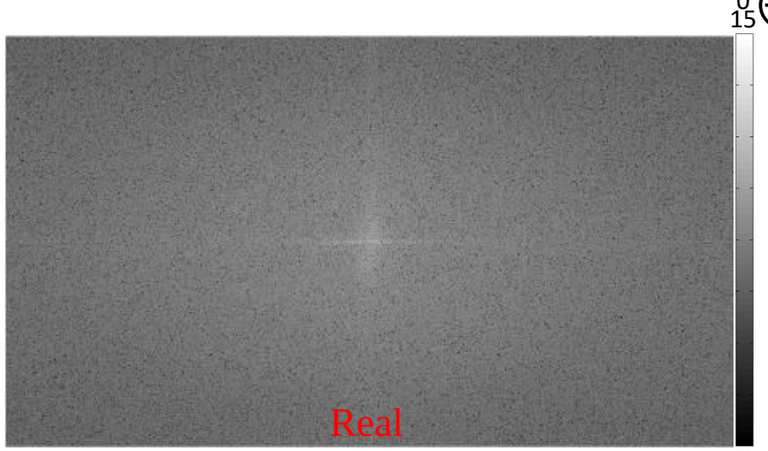


$\oplus$

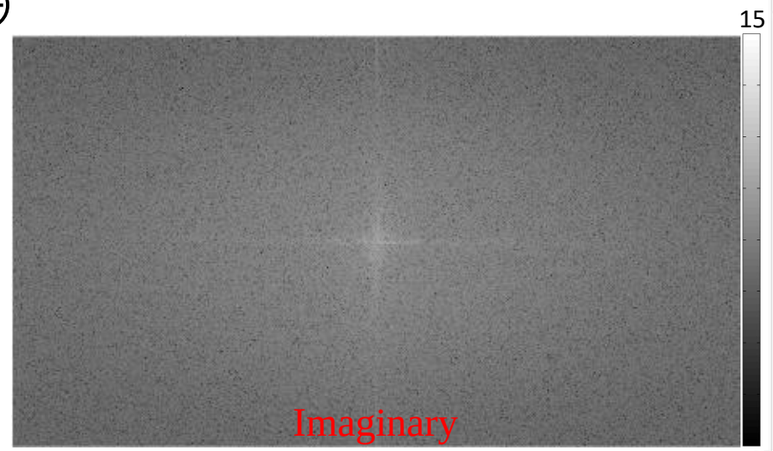


=

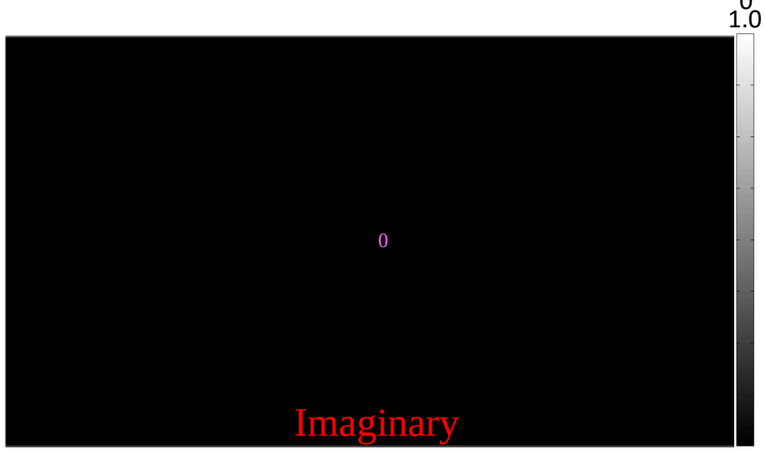




$\oplus$



=



*coefficients were log(abs(f)+1) for display

Template Matching via DFT

We now have all the pieces that we need.

$\sum o_i = 10358226$

$\sum o_i^2 = 1929989924$

$\sum p_i$

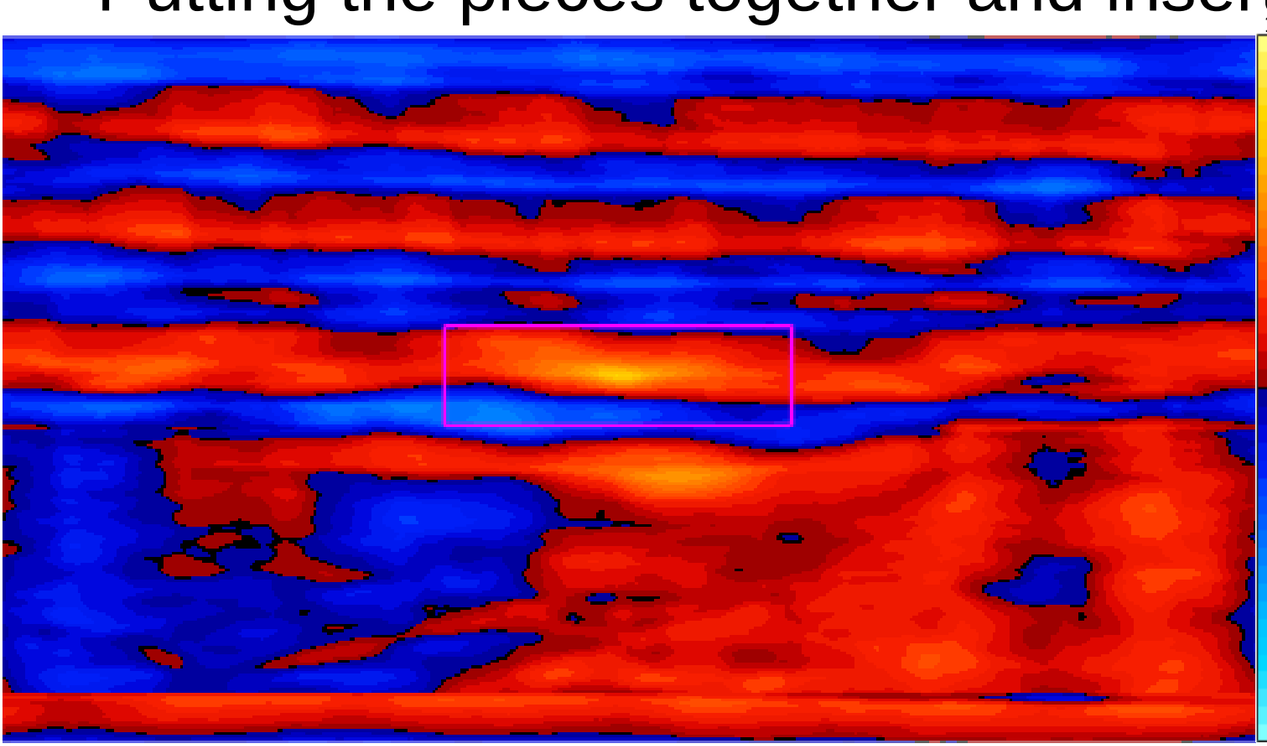
$\sum p_i^2$

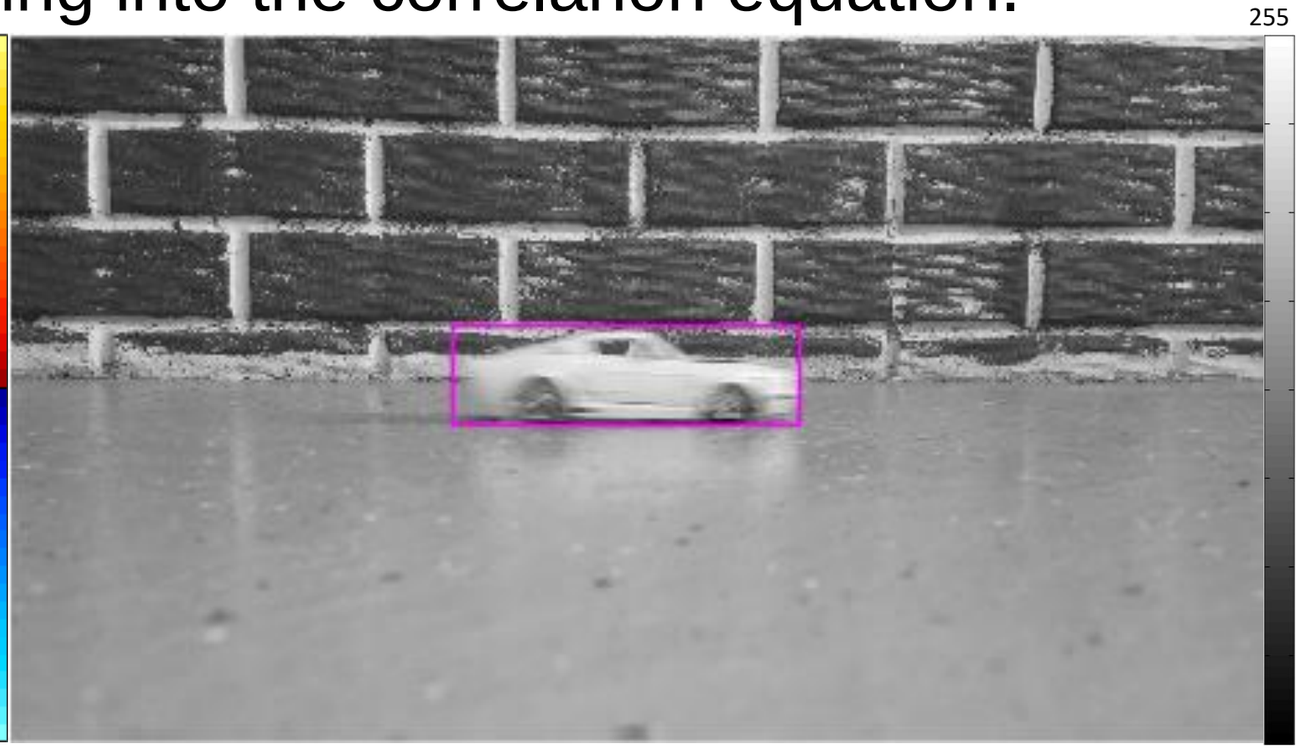
$\sum o_i p_i$

$$r = \frac{\sum p_i o_i - \frac{1}{n}\left(\sum p_i\right)\left(\sum o_i\right)}{\sqrt{\sum p_i^2 - \frac{1}{n}\left(\sum p_i\right)^2}\sqrt{\sum o_i^2 - \frac{1}{n}\left(\sum o_i\right)^2}}$$

Template Matching via DFT

Putting the pieces together and inserting into the correlarion equation.





$$r = \frac{\sum p_i o_i - \frac{1}{n}\left(\sum p_i\right)\left(\sum o_i\right)}{\sqrt{\sum p_i^2 - \frac{1}{n}\left(\sum p_i\right)^2}\sqrt{\sum o_i^2 - \frac{1}{n}\left(\sum o_i\right)^2}}$$

Template Matching via DFT

```
% extract and form template
gt500=squeeze(I(444:595,1318:1849,158));
figure;
imagesc(gt500)
colormap(gray), axis image, axis off
%imwrite(gt500,gray,'myGT.tif','Compression','none')
% edited in paint to make black background
myGT=double(imread('myGT.tif'));
gt500=myGT(:,:,1); myGT(:,:,2:4)=[];
figure;
imagesc(gt500)
colormap(gray), axis image, axis off
% template match for car one image%%%%%%%%%%%%%%%%%
% sample image of scene
t=140; It140=squeeze(I(:,:,t));
figure; % select one sample frame for convolution
imagesc(It140,[0,limMax])
colormap(gray), axis image, axis off

template=gt500;
[a,b]=size(template);
Sx=sum(template(:)); Sx2=sum(template(:).^2);
Varx=(Sx2-Sx^2/(a*b))/(a*b-1);
```

```
% generate ones kernel same size as template
tones=ones(a,b);
figure;
imagesc(tones,[0,1/25])
colormap(gray), axis image, axis off
line([0.5,b-.5],[.5,.5],'Color',...
    [0,0,0],'LineWidth',1)
line([0.5,b-.5],[a-.5,a-.5],'Color',....
    [0,0,0],'LineWidth',1)
line([0.5,.5],[.5,a-.5],'Color',...
    [0,0,0],'LineWidth',1)
line([b-.5,b-.5],[.5,a-.5],'Color',...
    [0,0,0],'LineWidth',1)
% place ones center of an image of zeros
tonesfill=zeros(ny,nx);
tonesfill(ny/2-a/2+1:ny/2+a/2,...
    nx/2-b/2+1:nx/2+b/2)=tones;
ftt1fill=fftshift(fft2(fftshift(tonesfill)));

% place template at center of zeros image
tfill=zeros(ny,nx); indx=961; indy=541;
tfill(ny/2-a/2+1:ny/2+a/2,...
    nx/2-b/2+1:nx/2+b/2)=template;
```

Template Matching via DFT

```
figure;
imagesc(tfill,[0,255])
colormap(gray), axis image, axis off
line([.5+indx-b/2,.5+indx-b/2],[.5+indy-a/2,.5+indy+a/2],'Color',[.2,.2,.2],'LineWidth',1.0)
line([.5+indx+b/2,.5+indx+b/2],[.5+indy-a/2,.5+indy+a/2],'Color',[.2,.2,.2],'LineWidth',1.0)
line([.5+indx-b/2,.5+indx+b/2],[.5+indy-a/2,.5+indy-a/2],'Color',[.2,.2,.2],'LineWidth',1.0)
line([.5+indx-b/2,.5+indx+b/2],[.5+indy+a/2,.5+indy+a/2],'Color',[.2,.2,.2],'LineWidth',1.0)
% calculate the FFT of the centered template
fttfill=fftshift(fft2(fftshift(tfill)));
% track in one sample image frame
% calculate the FFT of the image scene
t=140;
ftIt140=fftshift(fft2(fftshift(squeeze(I(:,:,t)))));
% image square for sample frame
figure;
imagesc(squeeze(I(:,:,t)).^2,[0,limMax^2])
colormap(gray), axis image, axis off
% calculate the FFT of the image square
ftI2t140=fftshift(fft2(fftshift((squeeze(I(:,:,t)).^2))));
% calculate the local sum(y) via FFT
Sy=real(fftshift(ifft2(fftshift(ftIt140.*ftt1fill))));
figure;
imagesc(real(Sy),[0,a*b*255])
axis image, colormap(gray), axis off
```

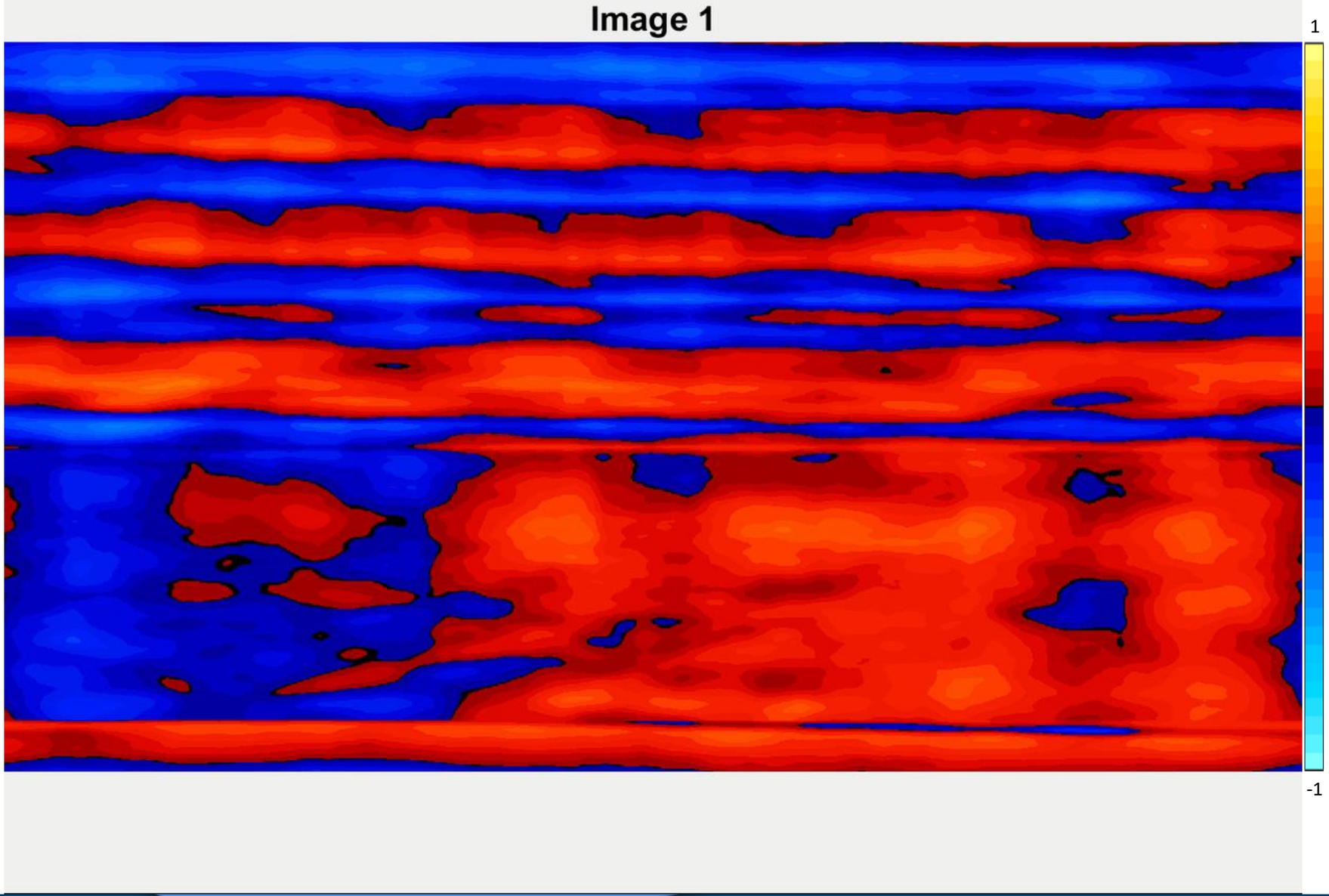
Template Matching via DFT

```
% calculate the local sum(y^2) via FFT
Sy2=real(fftshift(ifft2(fftshift(ftI2t140.*ftt1fill))));
figure;
imagesc(real(Sy2),[0,a*b*255^2])
axis image, colormap(gray), axis off
figure;
imagesc(imag(Sy2),[0,a*b*255^2])
axis image, colormap(gray), axis off
% calculate the local sum(xy) via FFT
Sxy=real(fftshift(ifft2(fftshift(ftIt140.*conj((fttfill))))));
figure;
maxSxy=max(max(real(Sxy(:))));
imagesc(real(Sxy),[0,maxSxy])
axis image, colormap(gray), axis off
figure;
imagesc(imag(Sxy),[0,1])
axis image, colormap(gray), axis off
% put it all together and compute correlation
Vary=(Sy2-(Sy.^2)/(a*b))/(a*b-1);
CovxyI140=(Sxy-(Sx*Sy)/(a*b))/(a*b-1);
CorxyI140=CovxyI140./sqrt((Varx)*(Vary));
```

Template Matching via DFT

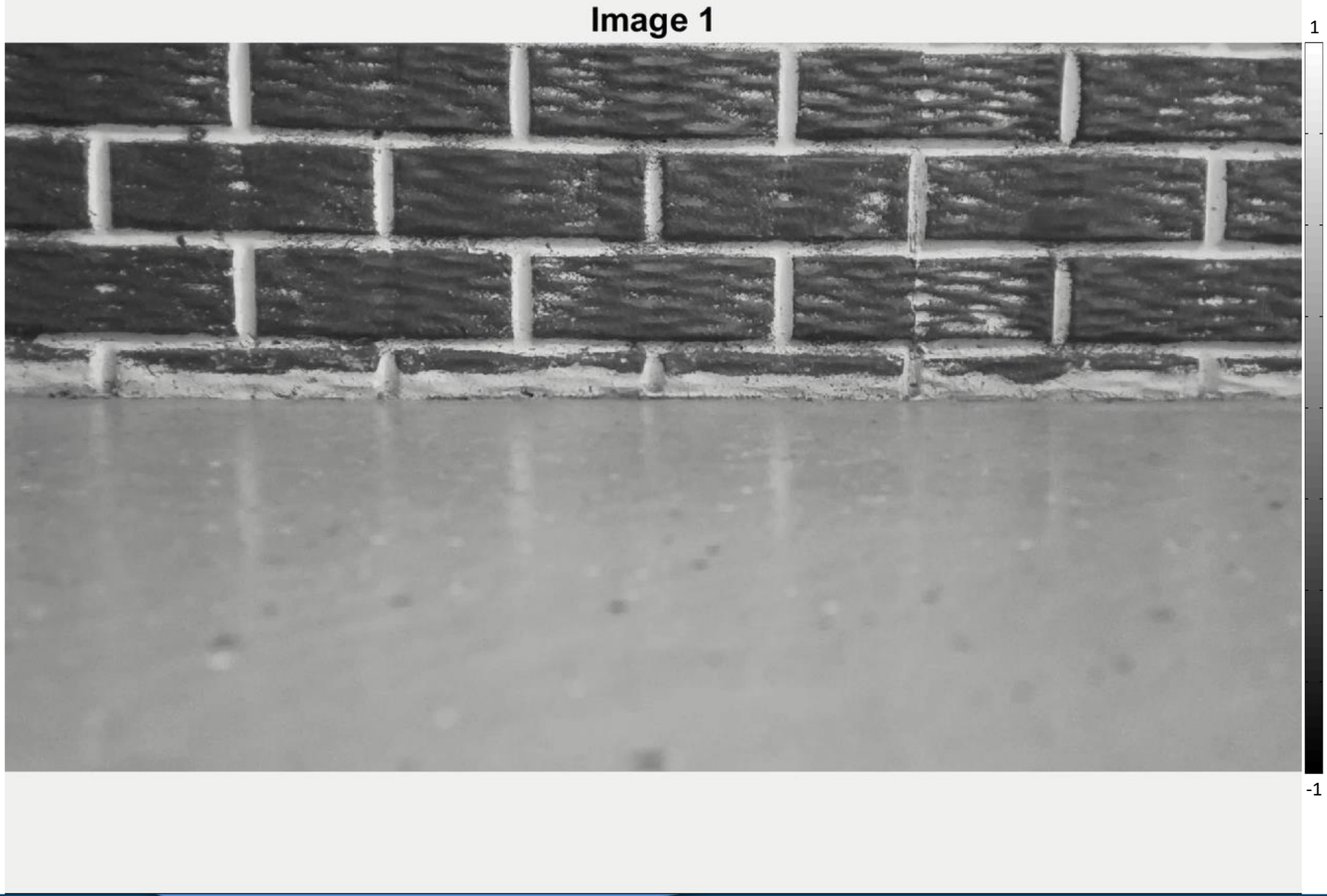
```
% find max val and indices for box
maxval=max(max(CorxyI140));
Cor140max(3,t)=maxval;
tmp=Cor140max(3,t);
[indy,indx]=find(CorxyI140==tmp);
Cor140max(1:2,t)=[indy,indx];
figure;
imagesc(CorxyI140,[-1,1])
axis image, colormap(myposnegmapblk), axis off
line([0.5+indx-b/2,0.5+indx-b/2],[0.5+indy-a/2,0.5+indy+a/2],'Color',[1,0,1],'LineWidth',1.0)
line([0.5+indx+b/2,0.5+indx+b/2],[0.5+indy-a/2,0.5+indy+a/2],'Color',[1,0,1],'LineWidth',1.0)
line([0.5+indx-b/2,0.5+indx+b/2],[0.5+indy-a/2,0.5+indy-a/2],'Color',[1,0,1],'LineWidth',1.0)
line([0.5+indx-b/2,0.5+indx+b/2],[0.5+indy+a/2,0.5+indy+a/2],'Color',[1,0,1],'LineWidth',1.0)
figure;
imagesc(It140,[0,limMax])
axis image, colormap(gray), axis off
line([0.5+indx-b/2,0.5+indx-b/2],[0.5+indy-a/2,0.5+indy+a/2],'Color',[1,0,1],'LineWidth',1.0)
line([0.5+indx+b/2,0.5+indx+b/2],[0.5+indy-a/2,0.5+indy+a/2],'Color',[1,0,1],'LineWidth',1.0)
line([0.5+indx-b/2,0.5+indx+b/2],[0.5+indy-a/2,0.5+indy-a/2],'Color',[1,0,1],'LineWidth',1.0)
line([0.5+indx-b/2,0.5+indx+b/2],[0.5+indy+a/2,0.5+indy+a/2],'Color',[1,0,1],'LineWidth',1.0)
```

Template Matching via DFT



Play Video TrackShelbyCor.avi

Template Matching via DFT



Play Video TrackShelby.avi

Discussion

We just saw how the DFT can be used to perform convolution.

With the DFT, we can smooth or sharpen images much faster
and quickly find objects in an image with template matching.

The FFT algorithm for the DFT makes real-time object tracking possible.

Discussion

Questions?

Homework 10

1. Use the DFT for convolution to perform template matching in an image of yours. Make images of I, I^2, centered kernel, centered template, their forward discrete Fourier transforms, the products of discrete forward Fourier transforms, inverse discrete Fourier Transforms, variance images, and correlation images. Comment.

2. Repeat #1 but for frames in a video where the object is moving.

3*. Track multiple objects in a video.

4*. Implement correlation yourself in image space for a small image. Compare times between image space and DFT space.

*For students in MSSC 5770.