# CGU Math 381 (Image Processing), Spring '18 – Homework #3

## Corner deteection, and an introduction to Fourier Analysis.

*Due on Friday 3/30, in the box in front of Prof. Micheli's office.*
*Song, Zhengming, Claremont Graduate University*

**Reading:** Fourier Analysis is treated in Chapters 7, 8 and 9 of *Principles of Digital Image Processing: Core Algorithms* (i.e. Volume 2) by Burger and Burge. The Harris corner detector is treated Chapter 4 of the same volume, and (in more detail) in the instructor's class notes, that are now available online.

Write, on top of the first page of your assignment: Name (LAST, First), your University or College, the HW#, and acknowledge other students with whom you may have worked (just write "*Worked with ...*"). For the computational problems where you are asked to write computer code you may choose the programming language that you prefer, such as *Python* (I am learning it myself!) or *Matlab*.

**Problem 3.1** (time shifting)**.** Consider the *shifted* delta function $\delta_{t_0}(t) = \delta(t - t_0)$. **(a)** Show that for any continuous-time signal $s(t)$, $t \in \mathbb{R}$, it is the case that $(s * \delta_{t_0})(t) = s(t - t_0)$ (that is, the convolution with $\delta_{t_0}$ hs the effect of time-shifting by $t_0$). **(b)** find the Fourier Transform $\mathcal{F}[\delta_{t_0}](f)$ by direct computation. **(c)** Finally, combine the above results with the convolution theorem (it states that $\mathcal{F}[g*h](f) = G(f)H(f)$) to find the Fourier Transform of the shifted signal $s(t - t_0)$ in terms of the Fourier Transform $S(f)$ of $s(t)$.

**Solution 3.1.** a

$$\begin{aligned}
(s * \delta_{t_0})(t) &= \int_{-\infty}^{+\infty} s(\tau)\delta_{t_0}(t-\tau)d\tau \\
&= \int_{-\infty}^{+\infty} s(\tau)\delta(t-t_0-\tau)d\tau \\
&= \int_{-\infty}^{+\infty} s(t-t_0)\delta(t-t_0-\tau)d\tau \\
&= s(t-t_0)\int_{-\infty}^{+\infty} \delta(t-t_0-\tau)d\tau \\
&= s(t-t_0)\int_{-\infty}^{+\infty} \delta(\tau-(t-t_0))d\tau \\
&= s(t-t_0)\cdot 1 \\
&= s(t-t_0)
\end{aligned}$$

b

$$\begin{aligned}
\mathcal{F}[\delta t_0](f) &= \int_{-\infty}^{+\infty} \delta_{t_0}(\tau)e^{-i2\pi f\tau}d\tau \\
&= \int_{-\infty}^{+\infty} \delta(\tau-t_0)e^{-i2\pi f\tau}d\tau \\
&\overset{\mathrm{t=\tau-t_0}}{=} \int_{-\infty}^{+\infty} \delta(t)e^{-i2\pi f(t+t_0)}dt \\
&= e^{-i2\pi t_0}\int_{-\infty}^{+\infty} \delta(t)e^{-i2\pi ft}dt \\
&= e^{-i2\pi t_0}\cdot e^{-i2\pi f0}\cdot\int_{-\infty}^{+\infty} \delta(t)dt \\
&= e^{-i2\pi t_0}\cdot 1\cdot 1 \\
&= e^{-i2\pi t_0}
\end{aligned}$$

c

$$\begin{aligned}
\mathcal{F}[s(t-t_0)](f) &= \mathcal{F}[s*\delta_{t_0}(t)](f) \\
&= \mathcal{S}(f)\mathcal{D}(f) \\
&= \mathcal{S}(f)\cdot\mathcal{F}[\delta_{t_0}](f) \\
&= e^{-i2\pi ft_0}\mathcal{S}(f)
\end{aligned}$$

**Problem 3.2** (the causal exponential)**.** Fix a constant $\alpha > 0$, and consider the continuous-time signal $s(t) = e^{-\alpha t}H(t)$, $t \in \mathbb{R}$, known as the *causal exponential* , where $H(t)$ is the Heaviside function (that is, $H(t) = 1$ for $t \geq 0$ and $H(t) = 0$ for $t < 0$). **(a)** Compute its Fourier transform $S(f)$, $f \in \mathbb{R}$ (this is example #26 in the Fourier pairs table that was distributed in class). **(b)** Compute the convolution $r(t) = (s * s)(t)$, $t \in \mathbb{R}$ (this may be the first and last time that you actually compute a convolution 'by hand', but it iss the kind of thing that you should do at least once in your life!). **(c)** Finally, using the result from part (a), what is the Fourier transform $R(f)$ of the continuous-time signal $r(t)$ from part (b)?

**Solution 3.2.** a

$$\begin{aligned}
\mathcal{S}(f) =& \quad \mathcal{F}[s](f) \\
=& \int_{-\infty}^{+\infty} e^{-\alpha t}H(t)e^{-i2\pi f t}dt \\
=& \int_{0}^{+\infty} e^{-\alpha t}e^{-i2\pi f t}dt \\
=& \int_{0}^{+\infty} e^{-\alpha + i2\pi f}tdt \\
=& \frac{e^{-(\alpha+i2\pi f)t}}{-(\alpha+i2\pi f)}\Big|_{0}^{+\infty} \\
=& \quad 0 - \frac{-1}{\alpha+i2\pi f} \\
=& \quad \frac{1}{\alpha+i2\pi f}
\end{aligned}$$

b

$$\begin{aligned}
r(t) =& \quad (s * s)(t) \\
=& \int_{-\infty}^{+\infty} s(\tau)s(t-\tau)d\tau \\
=& \int_{-\infty}^{+\infty} e^{-\alpha\tau}H(\tau)e^{-\alpha(t-\tau)}H(t-\tau)d\tau \\
=& \quad e^{-\alpha t}\int_{-\infty}^{+\infty} H(\tau)H(t-\tau)d\tau \\
=& \quad e^{-\alpha t}tH(t)
\end{aligned}$$

c

$$\mathcal{R}(f) = \mathcal{F}[r](f) = \mathcal{S}\cdot\mathcal{S}(f) = \frac{1}{(\alpha + i2\pi f)^2}$$

**Problem 3.3** (FT of separable 2D signals)**.** Suppose that a signal $s(x, y)$, with $(x, y) \in \mathbb{R}^2$, may be written as: $s(x, y) = f(x)g(y)$, where $f$ and $g$ are (continuous-space) signals. Such 2D signals are called *separable*. Show that the Fourier transform of $s$ is $S(u, v) = F(u)G(v)$, where $F(u) = \mathcal{F}[f](u)$ and $G(v) = \mathcal{F}[g](v)$.

**Solution 3.3.**

$$\begin{aligned}
\mathcal{S}(u,v) =& \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} s(x,y)e^{-i2\pi(ux+vy)}dxdy \\
=& \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} f(x)g(y)e^{-i2\pi(ux+vy)}dxdy \\
=& \int_{-\infty}^{+\infty} g(y)e^{-i2\pi vy}\int_{-\infty}^{+\infty} f(x)e^{-i2\pi ux}dxdy \\
=& \int_{-\infty}^{+\infty} g(y)e^{-i2\pi vy}\mathcal{F}(u)dy \\
=& \quad \mathcal{F}(u)\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} g(y)e^{-i2\pi vy}dy \\
=& \quad \mathcal{F}(u)\mathcal{G}(v)
\end{aligned}$$

**Problem 3.4** (differential operators and convolution)**.** Consider two continuous-space signals $f(x, y)$ and $g(x, y)$, with $(x, y) \in \mathbb{R}^2$. **(a)** Show that $\nabla^2(f * g) = f * (\nabla^2 g)$, where $\nabla^2$ donotes the Laplace operator in the variables $(x, y)$. **(b)** Express the Fourier transform of $\frac{\partial^2 f}{\partial x \partial y}$ in terms of $F(u, v) = \mathcal{F}[f](u, v)$.

**Solution 3.4.** a

$$\begin{aligned}\nabla^2(f*g)(x,y) = &\quad \nabla^2 \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} f(u,v)g(x-u,y-v)dudv\\ = &\quad \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} f(u,v)\cdot\nabla^2 g(x-u,y-v)dudv\\ = &\quad (f*\nabla^2 g)(x,y)\end{aligned}$$

b

$$\begin{aligned}\frac{\partial^2 f}{\partial x\partial y} = &\quad \tfrac{\partial}{\partial y}[\tfrac{\partial}{\partial x}(f)]\\ = &\quad \tfrac{\partial}{\partial y}[\tfrac{\partial}{\partial x}(\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} F(u,v)e^{i2\pi(ux+vy)}dudv)]\\ = &\quad \tfrac{\partial}{\partial y}[\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} F(u,v)\tfrac{\partial}{\partial x}e^{i2\pi(ux+vy)}dudv]\\ = &\quad \tfrac{\partial}{\partial y}[\int_{-\infty}^{+\infty} i2\pi u\int_{-\infty}^{+\infty} F(u,v)e^{i2\pi(ux+vy)}dudv]\\ = &\quad [\int_{-\infty}^{+\infty} i2\pi u\int_{-\infty}^{+\infty} F(u,v)\tfrac{\partial}{\partial y}e^{i2\pi(ux+vy)}dudv]\\ = &\quad [\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty}(i2\pi)^2 uvF(u,v)e^{i2\pi(ux+vy)}dudv]\end{aligned}$$

Hence,

$$\mathcal{F}[\frac{\partial^2 f}{\partial x\partial y}](u,v) = -4\pi^2 uvF(u,v)$$

**Problem 3.5** (symmetries)**.** Consider a continous-space signal $s(x,y),\ (x,y)\in\mathbb{R}^2$. Show that if the signal $s$ is real and has even symmetry, i.e. $s(x,y)=\overline{s(x,y)}=\overline{s(-x,-y)}=s(-x,-y)$, (where the bar indicates complex conjugation) then its Fourier transform $S(u,v)$ is also real, with even symmetry.

**Solution 3.5.**

$$\begin{aligned}\mathcal{S}(u,v) = &\quad \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} s(x,y)e^{-i2\pi(ux+vy)}dxdy\\ = &\quad \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} s(x,y)[\cos(2\pi(ux+vy)) - isin(2\pi(ux+vy))]dxdy\\ = &\quad \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} s(x,y)\cos(2\pi(ux+vy))dxdy + -i\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} s(x,y)\sin(2\pi(ux+vy))dxdy\\ = &\quad \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} s(x,y)\cos(2\pi(ux+vy))dxdy + 0\end{aligned}$$

The last equality hold because $\sin(\cdot)$ is a odd function, while $s(x,y)$ is a event function, hence, the product of those two functions are odd, and the integral of a odd function is 0. In addition, since both $s(x,y)$ and $\cos(\cdot)$ are real, $\mathcal{S}(u,v)$ is real.

$$\begin{aligned}\mathcal{S}(-u,-v) = &\quad \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} s(x,y)e^{i2\pi(-ux-vy)}dxdy\\ \overset{\text{x}=-\alpha,\text{y}=-\beta}{=} &\quad \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} s(x,y)e^{-i2\pi(u\alpha+v\beta)}d\alpha d\beta\\ = &\quad \mathcal{S}(u,v)\end{aligned}$$

**Problem 3.6** (UMF in the frequency domain)**.** Remember that the sharpening technique known as *unmask filtering* is defined as follows, for continuous-space images $f(x,y)$. First of all, a smoothing (low-pass) fiter, such as a Gaussian filter $h(x,y)=\frac{1}{2\pi\sigma^2}\exp\left(-\frac{1}{2}\frac{x^2+y^2}{\sigma^2}\right)$ is chosen (note that since $h$ has volume 1, we have $H(0,0)=1$). Then the smooth compunent of $f$ is computed via convolution: $m=h*f$. The sharp component of the image $f(x,y)$ (also known as the *unsharp mask*) is $s(x,y)=f(x,y)-m(x,y)$.

The sharpened image is then defined as $g(x, y) = f(x, y) + a \cdot s(x, y)$, where the constant $a > 0$ is the *sharpening strentgh*. For a generic smoothing convolutional kernel $h(x, y)$ with Fourier transform $H(u, v)$, show (as we did in class) that we can write, in frequency, $G(u, v) = L(u, v) \cdot F(u, v)$; express the frequency response $L(u, v)$ in terms of the function $H(u, v)$ and the constant $a$. Also, write an explicit expression for $L(u, v)$ when $h(x, y)$ is the Gaussian function given above, and plot the graph of $z = L(u, v)$ for one or two particular choices of $\sigma$ and $a$ (since $L(u, v)$ turns out to be a radial fucntion, i.e. such that its graph is symmetric with respect to rotations about the $z$-axis, you may just plot the cross section for $v = 0$).

**Solution 3.6.**

$$\begin{aligned} g(x,y) = & \quad f(x,y) + a \cdot s(x,y) \\ = & \quad f(x,y) + a \cdot f(x,y) - a \cdot h(x,y) * f(x,y) \end{aligned}$$

$$\mathcal{G}(u,v) = F(u,v) + aF(u,v) + aH(u,v)F(u,v) = (1 + a + aH(u,v))F(u,v)$$

Hence,

$$\mathcal{L}(u,v) = (1 + a + aH(u,v)) = (1 + a - ae^{-2\pi\sigma^2(u^2+v^2)})$$

```python
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from math import exp, pi
a = 0.8
sigma = 1
u = np.linspace(start=-1, stop=1, dtype=np.float32)
L = [1+a-a*exp(-2*pi*sigma**2*x**2) for x in u]
fig = plt.figure(figsize=(8, 6))
plt.plot(u,L)
plt.show()
fig.savefig('q6.png', dpi=fig.dpi)

```

Listing 1: Python Implementation of for Q6

**Problem 3.7** (DFT and IDFT)**.** In class we defined the Discrete Fourier Transform (DFT) of a discrete- and finite-time signal $s_n$, $n = 0, 1, \ldots, M-1$ as follows:

$$S_k = \sum_{n=0}^{M-1} s_n \, e^{-i2\pi kn/M} \qquad k = 0, 1, \ldots, M-1.$$

Show that $s_n$ can be recovered via the Inverse Discrete Fourier Transform: $s_n = \dfrac{1}{M} \displaystyle\sum_{k=0}^{M-1} S_k \, e^{+i2\pi kn/M}$.

*Hint:* You should first prove and then use the following *orthogonality* of discrete complex exponentials:

$$\sum_{k=0}^{M-1} e^{i2\pi km/M} e^{-i2\pi kn/M} = \begin{cases} M & \text{if } m = n \\ 0 & \text{otherwise.} \end{cases}$$

**Solution 3.7.** First let's prove the orthogonality of discrete complex exponential:

*Proof.*

$$\sum_{k=0}^{M-1} e^{i2\pi km/M} e^{-i2\pi kn/M} = \sum e^{2\pi ki(m-n)/M}$$

if $m = n$, we have

$$\sum e^{2\pi ki(m-n)/M} = \sum 1 = M$$

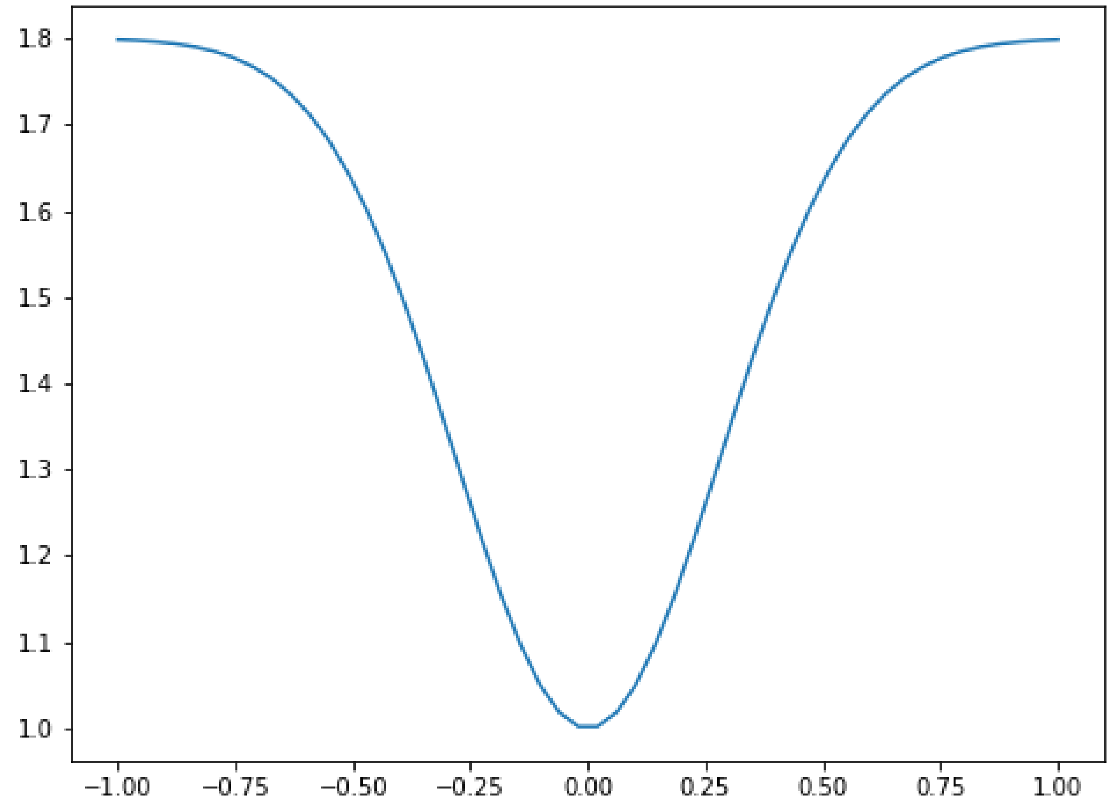


Figure 1: $\mathcal{L}(u,v)$ with $v=0$

if $m \neq n$, we have

$$\begin{aligned}\sum e^{2\pi k i(m-n)/M} &= \sum [e^{\frac{2\pi i(m-n)}{M}}]^k \\ &= \frac{1-(e^{\frac{2\pi i(m-n)}{M}})^M}{1-e^{\frac{2\pi i(m-n)}{M}}}\end{aligned}$$

Given that $m, n \in [0, 1, \cdots, M-1]$ hence, the denominator cannot be 0. however, in the numerator,

$$e^{2\pi i(m-n)} = \cos(2\pi(m-n)) + i\sin(2\pi(m-n)) = 1 + 0$$

, which makes the numerator 0.
Hence,

$$\sum_{k=0}^{M-1} e^{i2\pi km/M} e^{-i2\pi kn/M} = \begin{cases} M & \text{if } m = n \\ 0 & \text{otherwise.} \end{cases}$$

$\square$

Next, we proof the main statement of this problem

*Proof.*

$$
\begin{aligned}
s_n &= \frac{1}{M}\sum_{k=0}^{M-1} S_k e^{i2\pi kn/M} \\
&= \frac{1}{M}\sum_{k=0}^{M-1}\sum_{m=0}^{M-1} s_m e^{-i2\pi km/M} e^{i2\pi kn/M} \\
&= \frac{1}{M}\sum_{m=0}^{M-1} s_m \sum_{k=0}^{M-1} e^{-i2\pi km/M} e^{i2\pi kn/M} \\
&= \frac{1}{M}\cdot s_n \cdot M \\
&= s_n
\end{aligned}
$$

□

**Problem 3.8** (2D DFT of sine function)**.** Consider the 2D signal: $s_{m,n} = \sin(2\pi k_0 m + 2\pi \ell_0 m)$, with $m = 0, 1, 2, \ldots, M-1$ and $n = 0, 1, 2, \ldots, N-1$; also, $k_0 \in \{0, \frac{1}{M}, \frac{2}{M}, \ldots, \frac{M-1}{M}\}$ and $\ell_0 \in \{0, \frac{1}{N}, \frac{2}{N}, \ldots, \frac{N-1}{N}\}$ are constants. Show that its DFT is $S_{k\,\ell} = \frac{iMN}{2}\big[\delta(k + Mk_0, \ell + N\ell_0) - \delta(k - Mk_0, \ell - N\ell_0)\big]$, where $\delta$ is the 2D discrete delta function: $\delta(k,\ell) = \begin{cases} 1 & \text{for } k = \ell = 0, \\ 0 & \text{for } 1 \le k \le M-1 \text{ or } 1 \le k \le N-1. \end{cases}$

**Solution 3.8.**

$$
\begin{aligned}
S_{k,l} &= \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} s_{m,n} e^{-i2\pi(\frac{mk}{M}+\frac{nl}{N})} \\
&= \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} \sin(2\pi k_0 m + 2\pi l_0 n) e^{-i2\pi(\frac{mk}{M}+\frac{nl}{N})} \\
&= \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} [-\tfrac{i}{2}(e^{i(2\pi k_0 m + 2\pi l_0 n)} - e^{-i(2\pi k_0 m + 2\pi l_0 n)})] e^{-i2\pi(\frac{mk}{M}+\frac{nl}{N})} \\
&= \frac{i}{2}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} [e^{-i2\pi(k_0 m + l_0 n) - i2\pi(\frac{mk}{M}+\frac{nl}{N})} - e^{i2\pi(k_0 n + l_0 n) - i2\pi(\frac{mk}{M}+\frac{nl}{N})}] \\
&= \frac{i}{2}\{\sum_{m=0}^{M-1} e^{-i2\pi m(k_0+\frac{k}{M})} \sum_{n=0}^{N-1} e^{-i2\pi n(l_0+\frac{l}{N})} - \sum_{m=0}^{M-1} e^{i2\pi m(k_0-\frac{k}{M})} \sum_{n=0}^{N-1} e^{i2\pi n(l_0-\frac{l}{N})}\}
\end{aligned}
$$

if $Mk_0 + k = 0$, $\sum_{m=0}^{M-1} e^{-i2\pi m(k_0+\frac{k}{M})} = \sum_{m=0}^{M-1} 1 = M$, else,

$$
\begin{aligned}
\sum_{m=0}^{M-1} e^{-i2\pi m(k_0+\frac{k}{M})} &= \sum_{m=0}^{M-1} [e^{-i2\pi(k_0+\frac{k}{M})}]^m \\
&= \frac{1-(e^{-i2\pi(k_0+\frac{k}{M})})^M}{1-e^{e^{-i2\pi(k_0+\frac{k}{M})}}} \\
&= \frac{1-e^{-i2\pi(Mk_0+k)}}{1-e^{e^{-i2\pi(k_0+\frac{k}{M})}}} \\
&= \frac{1-(\cos(2\pi(Mk_0+k))-i\sin(2\pi(Mk_0+k)))}{1-e^{e^{-i2\pi(k_0+\frac{k}{M})}}} \\
&= 0
\end{aligned}
$$

Hence, we can write $\sum_{m=0}^{M-1} e^{-i2\pi m(k_0+\frac{k}{M})} = M\delta(k+Mk_0)$, $\sum_{n=0}^{N-1} e^{-i2\pi n(l_0+\frac{l}{N})} = N\delta(l+Nl_0)$, $\sum_{m=0}^{M-1} e^{i2\pi m(k_0-\frac{k}{M})} = M\delta(k - Mk_0)$, and $\sum_{n=0}^{N-1} e^{i2\pi n(l_0-\frac{l}{N})} = N\delta(l - Nl_0)$. Then, we have:

$$
S_{k\,\ell} = \frac{iMN}{2}\big[\delta(k + Mk_0, \ell + N\ell_0) - \delta(k - Mk_0, \ell - N\ell_0)\big]
$$

**Problem 3.9** (Harris corner detector[1])**.** In class we introduced the Harris corner detector:

$$
E_{(i,j)}(u,v) = \sum_{k=-\infty}^{\infty}\sum_{\ell=-\infty}^{\infty} w(k-i, \ell-j)\big[I(k,\ell) - I(k-u, \ell-v)\big]^2,
$$

[1] C. Harris and M. Stephens: A Combined Corner and Edge Detector. In *Proceedings of the Fourth Alvey Vision Conference*, pages 147–151. University of Manchester, August 31—September 2, 1988.

which measures the change of the image $I$ around the location $(i,j)$ due to a small displacement $(u,v)$. Typically, $w$ is a $(2K+1)\times(2K+1)$ filter of ones centered around the origin, but is could also be a rotationally invariant Gaussian mask. In class we showed[2] that we can approximate $(\star)$ with

$$E_{(i,j)}(u,v) = \begin{bmatrix} u & v \end{bmatrix} \cdot \overline{M}(i,j) \cdot \begin{bmatrix} u \\ v \end{bmatrix},$$

where
$$\overline{M}(i,j) = \sum_{k=-\infty}^{\infty} \sum_{\ell=-\infty}^{\infty} w(k-i,\ell-j) \begin{bmatrix} I_x^2(k,\ell) & I_x(k,\ell)I_y(k,\ell) \\ I_x(k,\ell)I_y(k,\ell) & I_y^2(k,\ell) \end{bmatrix}$$

is called the *local structure matrix*. We saw that we have evidence of the existence of a corner at $(i,j)$ when both eigenvalues $\lambda_1(i,j)$ and $\lambda_2(i,j)$ of $\overline{M}(i,j)$ are large, and of comparable magnitude. This happens when the so-called *corner response function*

$$\boxed{Q_\alpha(i,j) = \det\left(\overline{M}(i,j)\right) - \alpha\left[\mathrm{Tr}\left(\overline{M}(i,j)\right)\right]^2}$$

is largest (typically the parameter $\alpha$ is chosen somewhere in the range between 0.04 and 0.15; this is discussed at the very end of the class notes).

**(a)** Choose $\alpha = 0.10$, and produce a contour plot of the corner response function with respect the eigenvalues (i.e., of $Q = \lambda_1\lambda_2 - \alpha(\lambda_1+\lambda_2)^2$). You should get a plot similar to Fig. 5 of Harris' paper[1]. (Feel free to use *Mathematica* , as it has one-line commands that allow you to achieve this.)

**(b)** Now download the images `image-polygons.gif`, `image-house.gif`, *and* other images of your own choice. Write code that computes the function $Q_\alpha(i,j)$, and plot, on top the image, markers (e.g. red crosses or bullets) that correspond to points within the image where $Q_\alpha$ is above a threshold $t$ of your choice. *Remark:* There are several paramenters that you can play with. For my own code, I have used a $9\times 9$ filter $w$ of ones, $\alpha = 0.05$, and a threshold for $Q_\alpha$ of $t = 10^7$.

**Solution 3.9.** a

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
alpha = 0.10
l1 = np.linspace(start=0, stop=0.5, dtype=np.float32)
l2 = np.linspace(start=0, stop=0.5, dtype=np.float32)
lambda1,lambda2 = np.meshgrid(l1, l2)
Q = lambda1*lambda2 - alpha*np.power(lambda1+lambda2,2)
fig = plt.figure(figsize=(8, 6))
plt.contour(lambda1, lambda2, Q)
plt.show()
fig.savefig('q9.png', dpi=fig.dpi)

```

Listing 2: Plot the corner response function

b

```
def getHarrisCornerResponse(I, h, alpha, threshold=0.8):
    '''
    This function calculate the Harris corner response funciton

    Inputs:
    * I: 2*2 array of Image with grey levels
    * h: filter, uniform, gaussian, etc.
    * alpha: parameter of the corner response equation
```

[2] The fact that $(u,v)$ is small allowed us to use a first-order Taylor expansion for the term in square brackets in $(\star)$.

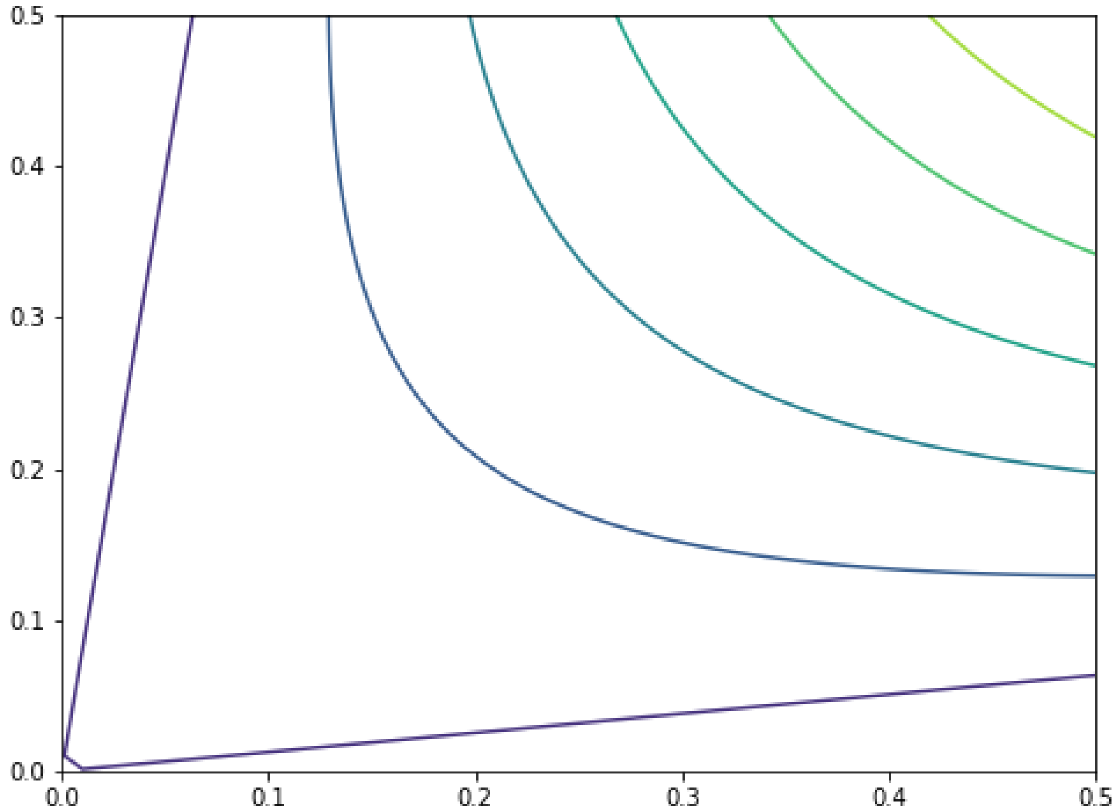


Figure 2: Plot of the corner response function

```
        * threshold: if <1, apply threshold based on the maximum of corner
response value, otherwise, directly apply input value

        Outputs:
        * Q: Matrix container response value
        * coordinates:
            tuple contains two arrays,
                first array is the x coordiate of the potins has corner
response value larger than threshold
                second array is the y coordiate of the potins has corner
response value larger than threshold
        '''
        (M,N) = I.shape
        hx = np.array([[0,0.5,0],[0,0,0],[0,-0.5,0]])
        Ix = scipy.ndimage.filters.convolve(I,hx)
        Ix2 = np.power(Ix, 2)
        hy = hx.transpose()
        Iy = scipy.ndimage.filters.convolve(I,hy)
        Iy2 = np.power(Iy, 2)
        IxIy =  scipy.ndimage.filters.convolve(Ix,hy)
        ul =  scipy.ndimage.filters.convolve(Ix2,h)
        ur =  scipy.ndimage.filters.convolve(IxIy,h)
        bl =  scipy.ndimage.filters.convolve(IxIy,h)
```

```
        br = scipy.ndimage.filters.convolve(Iy2,h)
        Q_alpha = np.zeros((M,N))
        for i_ind in range(M):
            for j_ind in range(N):
                m_bar = np.array([[ul[i_ind,j_ind], ur[i_ind,j_ind]],[bl[i_ind,j_ind],br[i_ind,j_ind]]])
                Q_alpha[i_ind,j_ind] = np.linalg.det(m_bar) - alpha*(np.trace(m_bar))**2
        if threshold<=1:
            t = np.max(Q_alpha)*threshold
        else:
            t= threshold
        coordinates = np.where(Q_alpha>t)
        return Q_alpha, coordinates

```

Listing 3: Implementation of function $Q_a$

```
n = 3
w = np.ones(shape=(n,n))
h = w
Q,coordinates = getHarrisCornerResponse(I,h,0.05, 0.03)
fig=plt.figure(1,figsize=(M/20,N/20))
plt.imshow(I,cmap=cm.gray,vmin=0,vmax=255)
plt.scatter(coordinates[1],coordinates[0],c='r', s=5)
plt.axis('off')
fig.savefig('q9_house_uniform_filter.png', dpi=fig.dpi)

```

Listing 4: house with uniform filter

```
def gaussian(x,y,sigma):
    return exp(-(x**2+y**2)/(2*sigma**2))
n = 5
sigma=1
_hG = np.array([[ gaussian(r, c, sigma) for c in range(int(-(n-1)/2),int((n-1)/2 + 1))]
                for r in range(int(-(n-1)/2),int((n-1)/2 + 1))])
hG = _hG/np.sum(_hG)
Q,coordinates = getHarrisCornerResponse(I,hG,0.05, 0.01)
fig=plt.figure(1,figsize=(M/20,N/20))
plt.imshow(I,cmap=cm.gray,vmin=0,vmax=255)
plt.scatter(coordinates[1],coordinates[0],c='r', s=5)
plt.axis('off')
fig.savefig('q9_house_gaussian_filter.png', dpi=fig.dpi)

```

Listing 5: house with gaussian filter

```
n = 3
w = np.ones(shape=(n,n))
h = w
Q,coordinates = getHarrisCornerResponse(I,h,0.05, 1e7)
fig=plt.figure(1,figsize=(M/20,N/20))
plt.imshow(I,cmap=cm.gray,vmin=0,vmax=255)
plt.scatter(coordinates[1],coordinates[0],c='r', s=5)
plt.axis('off')
fig.savefig('q9_polygons_uniform_filter.png', dpi=fig.dpi)

```

Listing 6: polygon with uniform filter

```python
        n = 5
        sigma=1
        _hG = np.array([[ gaussian(r, c, sigma) for c in range(int(-(n-1)/2),int((n-1)/2 + 1))]
                        for r in range(int(-(n-1)/2),int((n-1)/2 + 1))])
        hG = _hG/np.sum(_hG)
        Q,coordinates = getHarrisCornerResponse(I,hG,0.05, 0.1)
        fig=plt.figure(1,figsize=(M/20,N/20))
        plt.imshow(I,cmap=cm.gray,vmin=0,vmax=255)
        plt.scatter(coordinates[1],coordinates[0],c='r', s=5)
        plt.axis('off')
        fig.savefig('q9_polygons_gaussian_filter.png', dpi=fig.dpi)
```

Listing 7: polygon with gaussian filter

```python
        n = 3
        w = np.ones(shape=(n,n))
        h = w
        Q,coordinates = getHarrisCornerResponse(I,h,0.05, 0.05)
        fig=plt.figure(1,figsize=(M/20,N/20))
        plt.imshow(I,cmap=cm.gray,vmin=0,vmax=255)
        plt.scatter(coordinates[1],coordinates[0],c='r', s=5)
        plt.axis('off')
        fig.savefig('q9_uae_uniform_filter.png', dpi=fig.dpi)
```

Listing 8: grand mosque with uniform filter

```python
        n = 5
        sigma=1
        _hG = np.array([[ gaussian(r, c, sigma) for c in range(int(-(n-1)/2),int((n-1)/2 + 1))]
                        for r in range(int(-(n-1)/2),int((n-1)/2 + 1))])
        hG = _hG/np.sum(_hG)
        Q,coordinates = getHarrisCornerResponse(I,hG,0.05, 0.1)
        fig=plt.figure(1,figsize=(M/20,N/20))
        plt.imshow(I,cmap=cm.gray,vmin=0,vmax=255)
        plt.scatter(coordinates[1],coordinates[0],c='r', s=5)
        plt.axis('off')
        fig.savefig('q9_uae_gaussian_filter.png', dpi=fig.dpi)
```

Listing 9: grand mosque with gaussian filter

Figure 3: house with uniform filter

Figure 4: house with gaussian filter

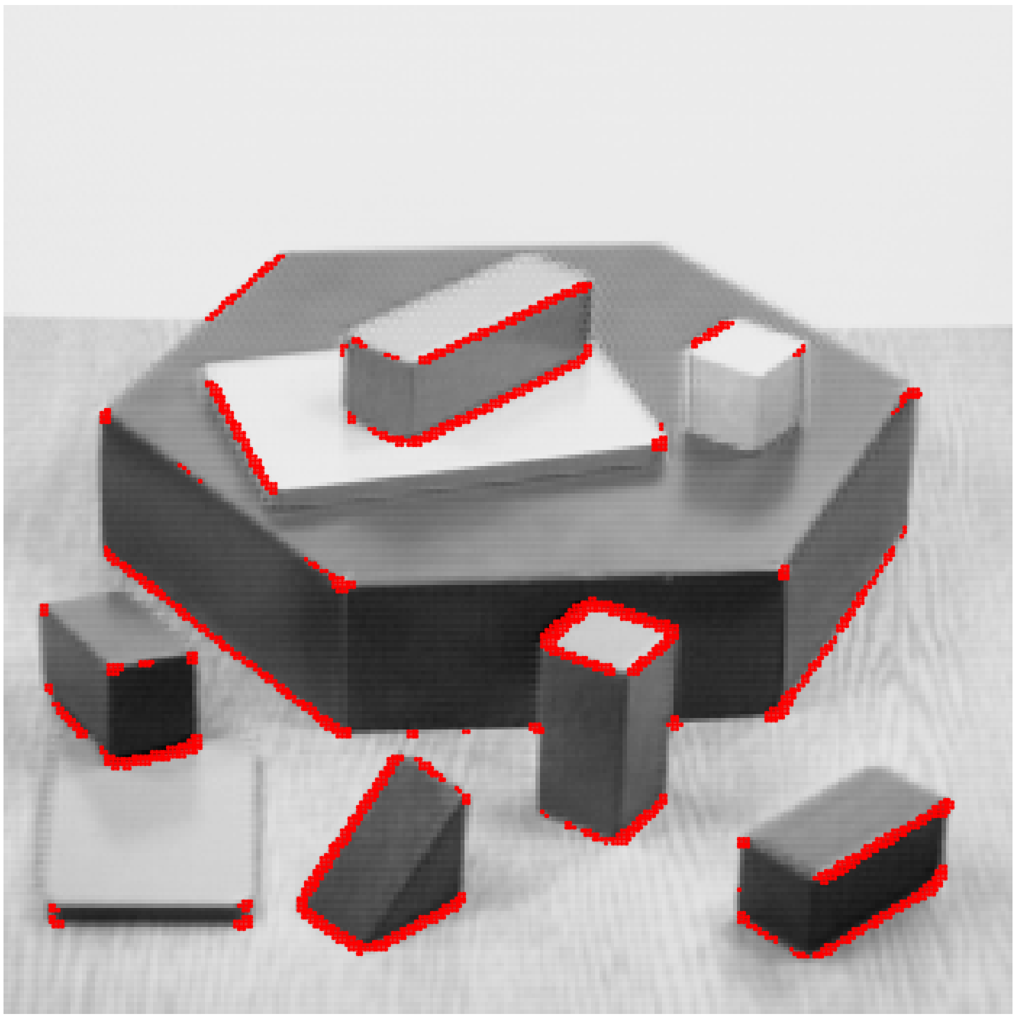

Figure 5: polygon with uniform filter

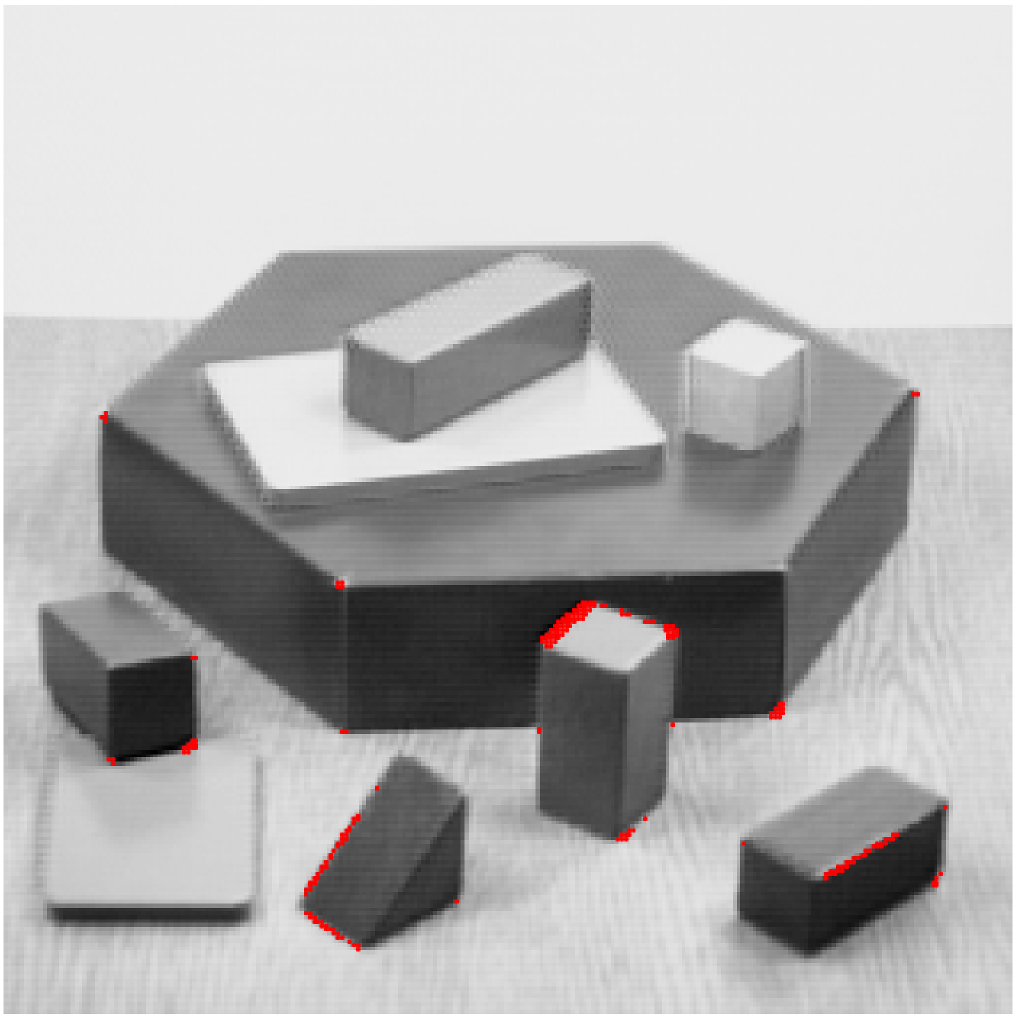

Figure 6: polygon with gaussian filter