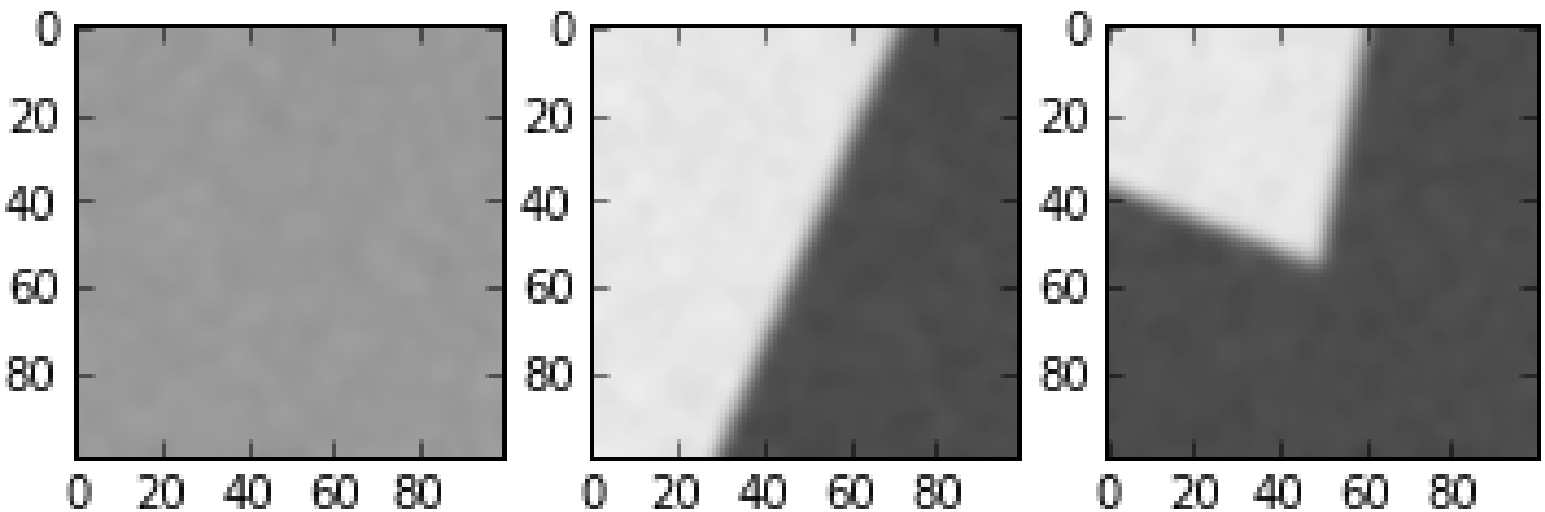
Sample Ideal Corner Images

```
ys,xs = mgrid[:100,:100]
flat = clip(G(0.2*rand(100,100)+0.5,2),0,1)
edge = clip(G(0.2*rand(100,100)+0.2+0.6*(0.3*ys+0.7*xs<50),2),0,1)
corner = clip(G(0.2*rand(100,100)+0.2+0.6*(0.1*ys+0.5*xs<30)*(0.8*
    ys-0.3*xs<30),2),0,1)
subplot(131); imshow(flat,vmin=0,vmax=1)
subplot(132); imshow(edge,vmin=0,vmax=1)
subplot(133); imshow(corner,vmin=0,vmax=1)
```



Idea behind Harris

Corners / interest points

- corners are places where edges of different orientations meet
- interest points are places that can be detected stably

Stability

Stable corner locations

Let's define stability as the difference between the shifted image and the image, averaged over a window w .

$$S(x,y) = \sum_u \sum_v w(u,v)\left(I(u+x, v+y) - I(x,y)\right)^2$$

$I(u+x, v+y)$ can be approximated by a Taylor series:

$$I(u+x, v+y) \approx I(u,v) + I_x(u,v)x + I_y(u,v)y$$

This then gives rise to:

$$S(x,y) \approx \sum_u \sum_v w(u,v)\left(I_x(u,v)x + I_y(u,v)y\right)^2$$

Matrix expression

$$S(x,y) \approx \sum_u \sum_v w(u,v)\ (I_x(u,v)x + I_y(u,v)y)^2$$

This can be written in matrix form:

$$S(x,y) \approx \begin{pmatrix} x & y \end{pmatrix} A \begin{pmatrix} x \\ y \end{pmatrix}$$

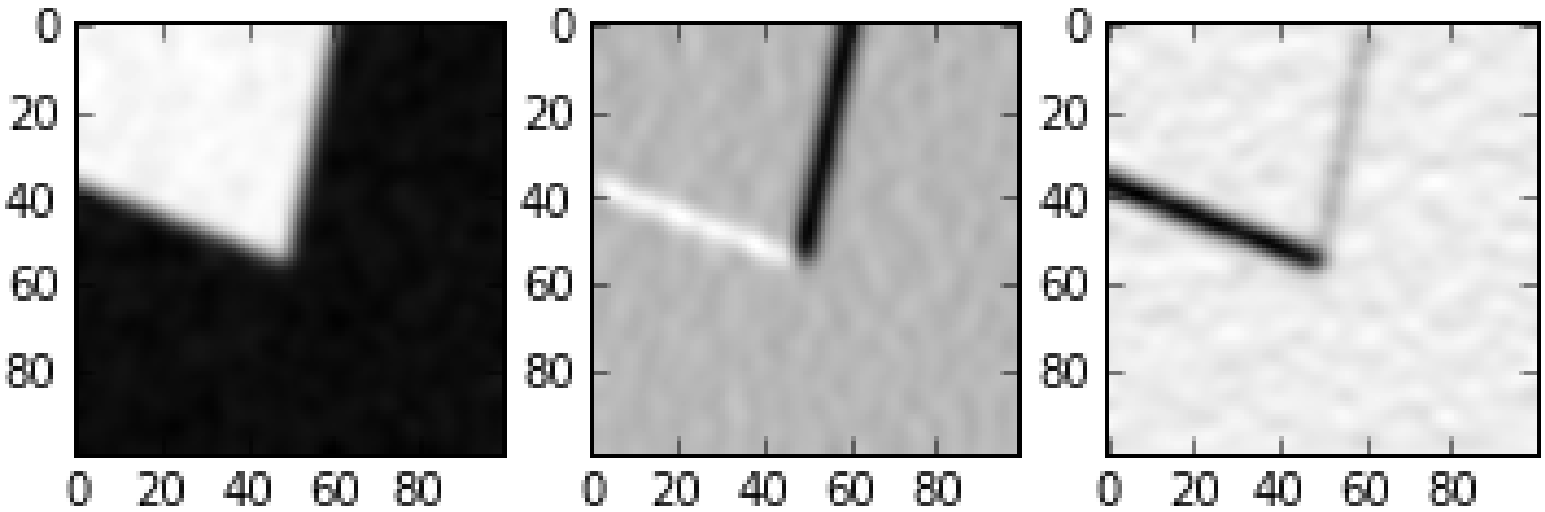
$$A = \sum_u \sum_v w(u,v) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

That derivation is fine, but another derivation is perhaps more straightforward.

Diversity of Gradient Orientations

gradient computation

```
dx = filters.gaussian_filter(corner,1,(0,1))
dy = filters.gaussian_filter(corner,1,(1,0))
subplot(131); imshow(corner)
subplot(132); imshow(dx)
subplot(133); imshow(dy)
```

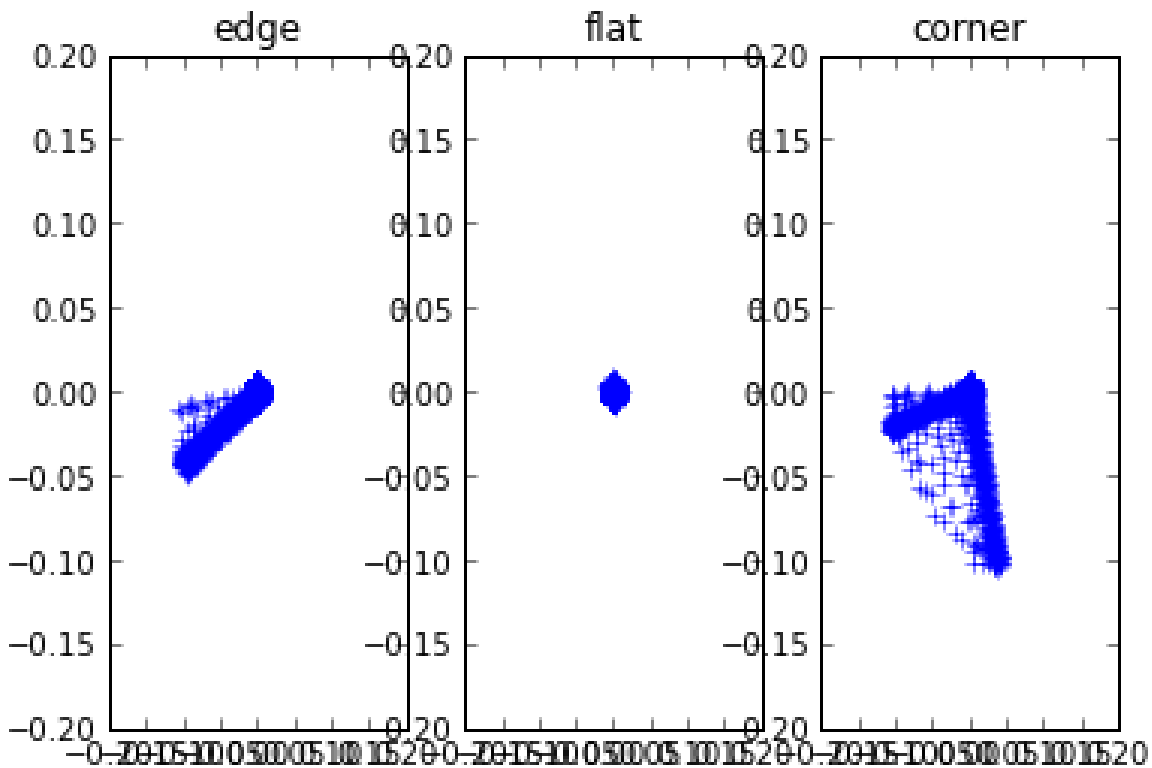


gradient histograms

```
dx = filters.gaussian_filter(flat,1,(0,1))
dy = filters.gaussian_filter(flat,1,(1,0))
fgs = c_[dx.ravel(),dy.ravel()]
dx = filters.gaussian_filter(edge,1,(0,1))
dy = filters.gaussian_filter(edge,1,(1,0))
egs = c_[dx.ravel(),dy.ravel()]
dx = filters.gaussian_filter(corner,1,(0,1))
dy = filters.gaussian_filter(corner,1,(1,0))
cgs = c_[dx.ravel(),dy.ravel()]
```

gradient histograms

```
1 subplot(131); xlim((-0.2,0.2)); ylim((-0.2,0.2)); plot(egs[:,0],egs
      [:,1],'b+'); title("edge")
2 subplot(132); xlim((-0.2,0.2)); ylim((-0.2,0.2)); plot(fgs[:,0],fgs
      [:,1],'b+'); title("flat")
3 subplot(133); xlim((-0.2,0.2)); ylim((-0.2,0.2)); plot(cgs[:,0],cgs
      [:,1],'b+'); title("corner")
```



```
print eig(dot(fgs.T,fgs))[0]
print eig(dot(egs.T,egs))[0]
print eig(dot(cgs.T,cgs))[0]
```

```
[ 0.05539413  0.0514928 ]
[ 4.91494968  0.07114731]
[ 2.77089285  2.13992565]
```

The matrix we are computing the eigenvalues of is similar to the covariance matrix (we already know that the "mean" should be at (0,0))

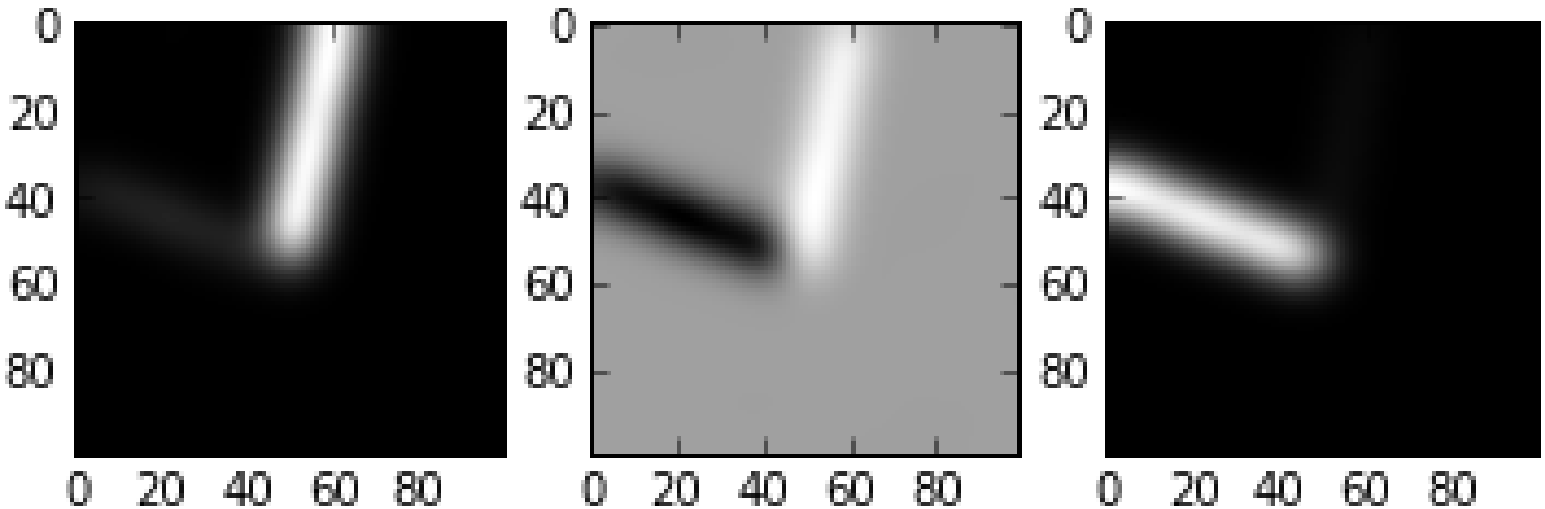
$$A = \left(\sum_i (g_i^x)^2 \sum_i g_i^x g_i^y \right.$$

$\sum_i g_i^x g_i^y \sum_i (g_i^y)^2$

Computing this over a patch is the same as summing g^x and g^y locally with a uniform filter. However, we can generalize this and use a Gaussian filter, giving smoother estimates.

```
1 dx = filters.gaussian_filter(corner,1,(0,1))
2 dy = filters.gaussian_filter(corner,1,(1,0))
3 sxx = filters.gaussian_filter(dx**2,5.0)
4 sxy = filters.gaussian_filter(dx*dy,5.0)
5 syy = filters.gaussian_filter(dy**2,5.0)
```

```
1 subplot(131); imshow(sxx)
2 subplot(132); imshow(sxy)
3 subplot(133); imshow(syy)
```



Eigenvalue computation

The eigenvalue computation requires solving:

$$(A - \lambda I)\, v = 0$$

In the 2×2 case:

$$\det\left(\lambda - a - b\right.$$

-c $\lambda - d = 0$

This gives rise to the quadratic equation:

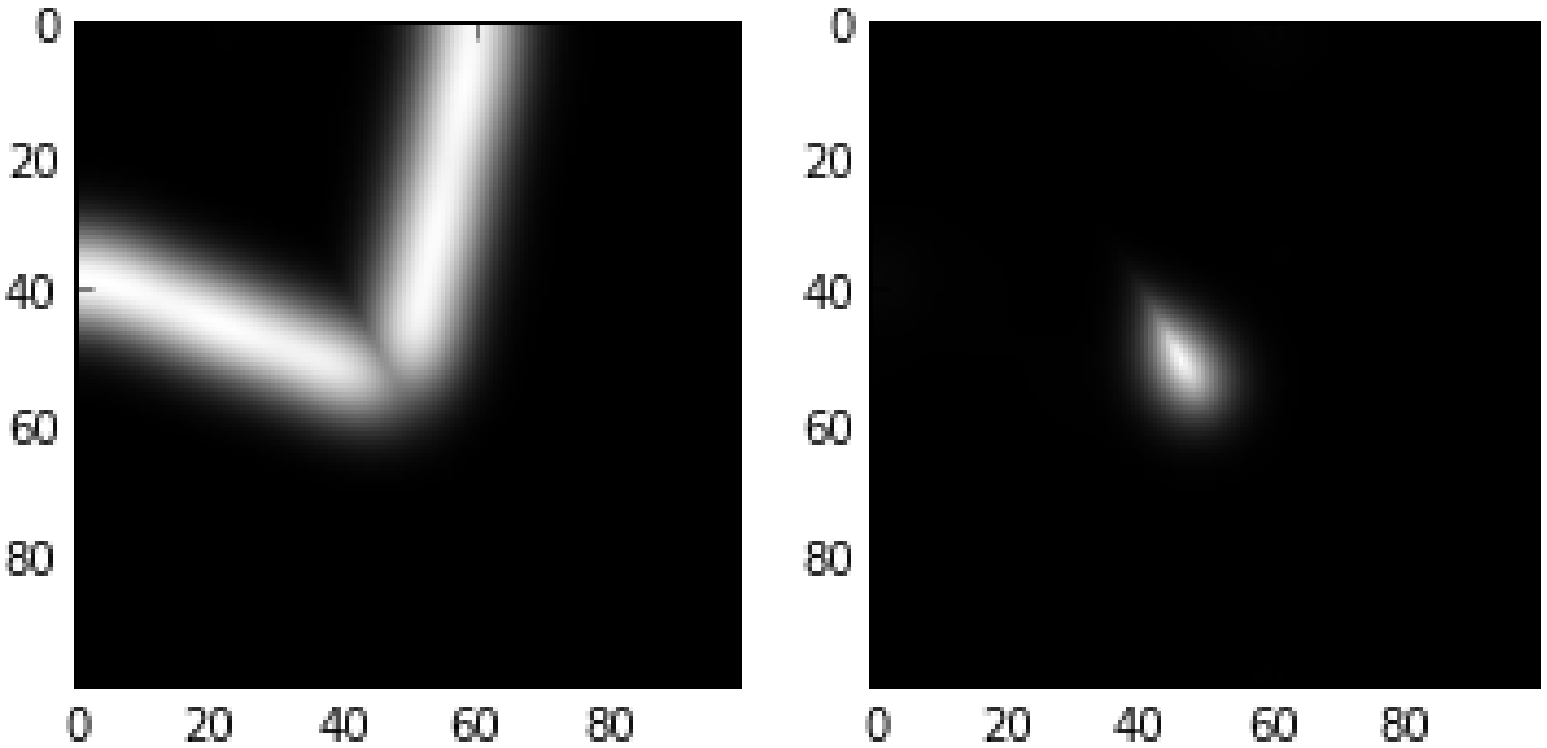
$$\lambda^2 - (a + d)\,\lambda + (ad - bc) = \lambda^2 - \lambda \mathrm{tr}(A) + \det(A) = 0$$

The solution is given by:

$$\lambda = \frac{\mathrm{tr}(A) \pm \sqrt{\mathrm{tr}^2(A) - 4\det(A)}}{2}$$

Eigenvalue computation

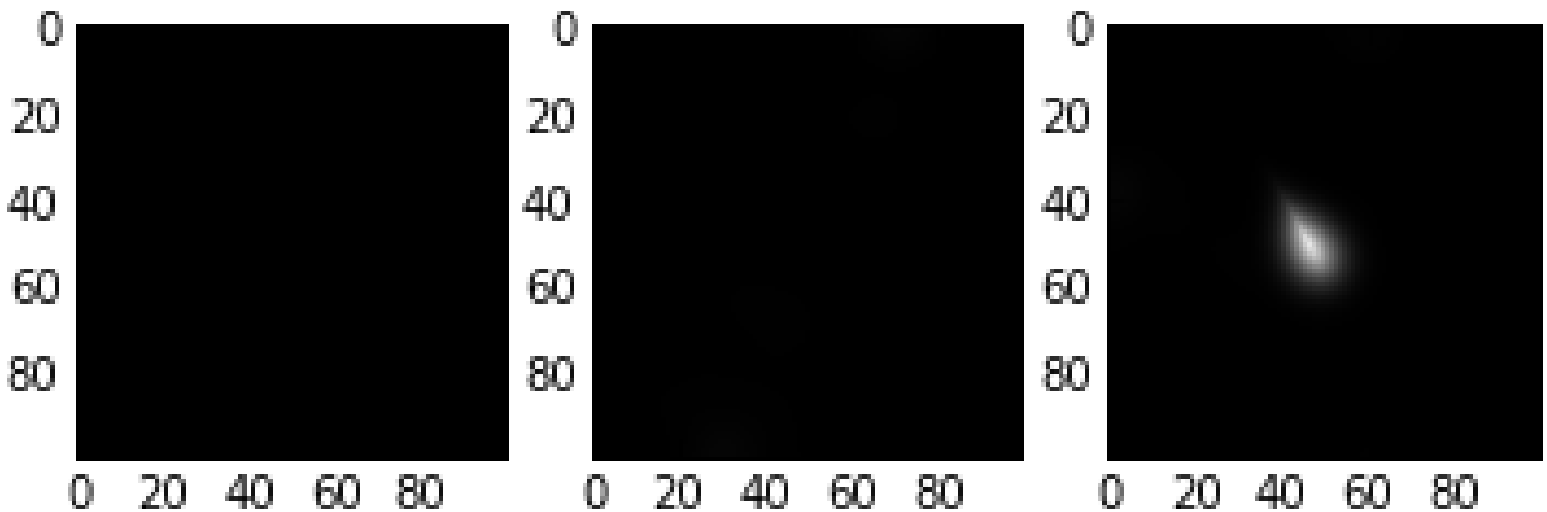
```
T = sxx+syy
D = sxx*syy-sxy**2
R = (T**2-4*D)**.5
hi = (T+R)/2
lo = (T-R)/2
subplot(121); imshow(hi)
subplot(122); imshow(lo)
```



```
def harris(image,sigma=1.0,area=5.0):
    dx = filters.gaussian_filter(image,sigma,(0,1))
    dy = filters.gaussian_filter(image,sigma,(1,0))
    sxx = filters.gaussian_filter(dx**2,area)
    sxy = filters.gaussian_filter(dx*dy,area)
    syy = filters.gaussian_filter(dy**2,area)
    T = sxx+syy
    D = sxx*syy-sxy**2
    R = (T**2-4*D)**.5
    hi = (T+R)/2
    lo = (T-R)/2
    return lo
```

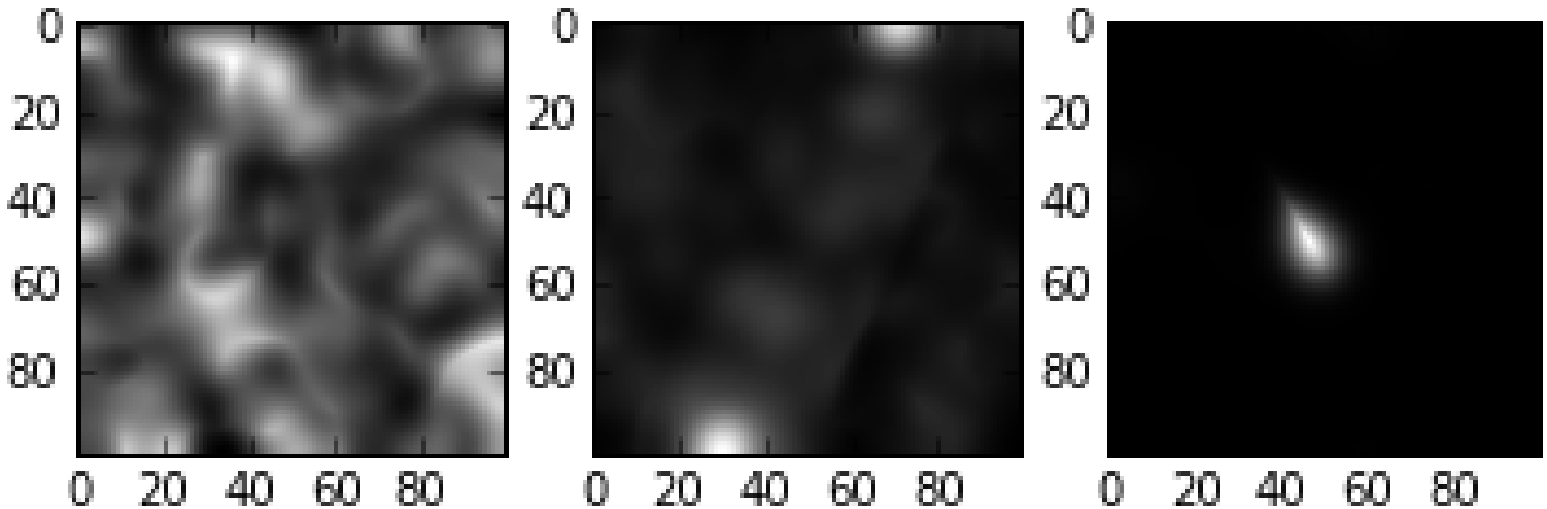
Harris corner detector

```
1 subplot(131); imshow(harris(flat),vmax=0.002)
2 subplot(132); imshow(harris(edge),vmax=0.002)
3 subplot(133); imshow(harris(corner),vmax=0.002)
```

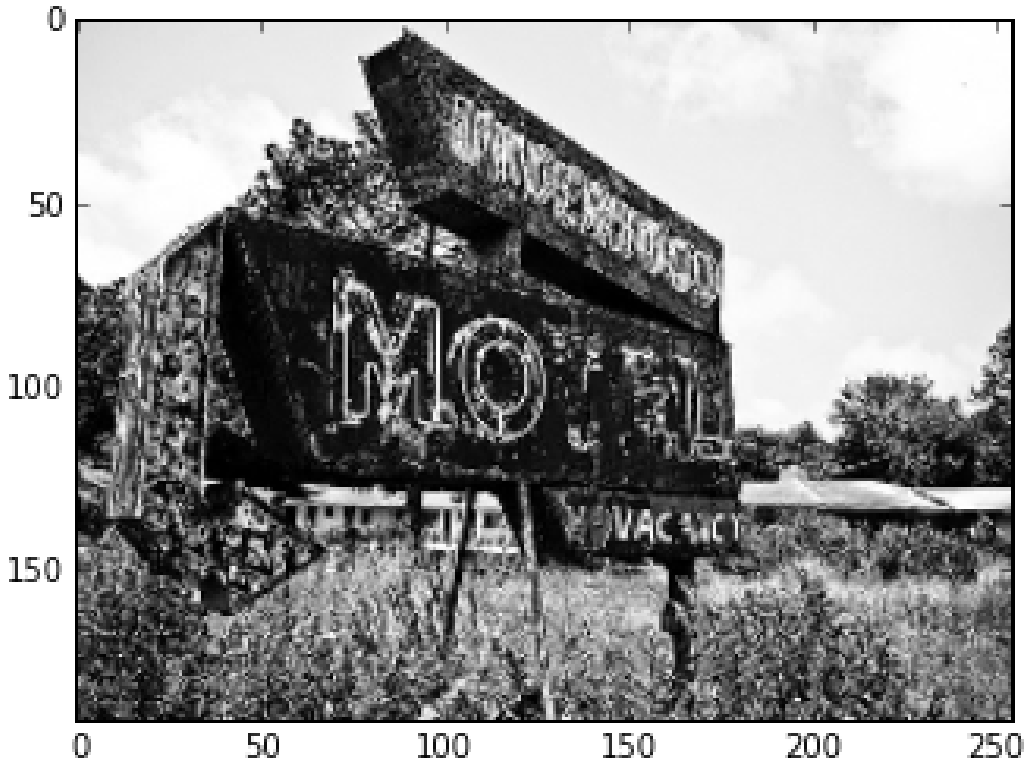


Harris corner detector (relative corners)

```
1 subplot(131); imshow(harris(flat))
2 subplot(132); imshow(harris(edge))
3 subplot(133); imshow(harris(corner))
```



```
import tables
hdf = tables.openFile("1k.h5")
images = hdf.root.icons
image = mean(crop_black(images[77]),2)
imshow(image)
```



```
1 imshow(harris(image,sigma=2.0,area=7.0))
```