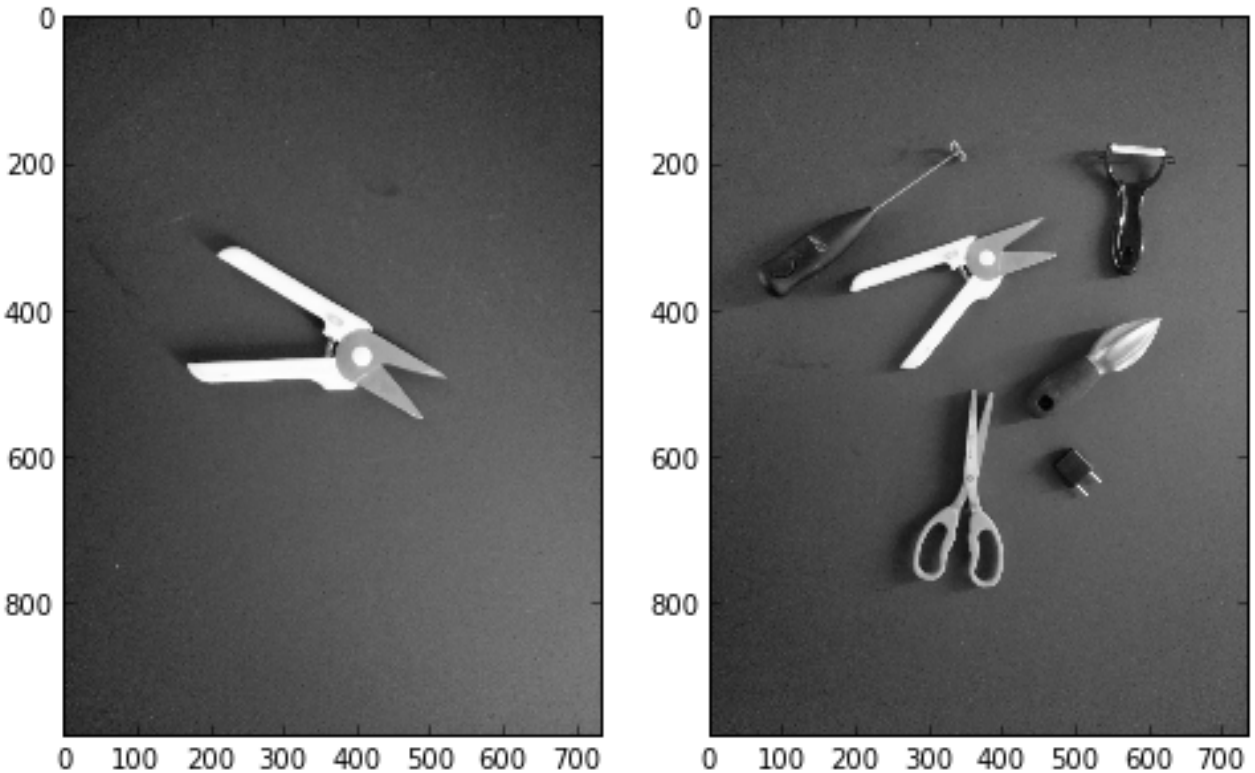
Sample Images

```
1 target = interpolation.zoom(mean(imread("match-target.png"),axis=2)
      ,0.3)
2 subplot(121); imshow(target)
3 clutter = interpolation.zoom(mean(imread("match-clutter.png"),axis
      =2),0.3)
4 subplot(122); imshow(clutter)
```

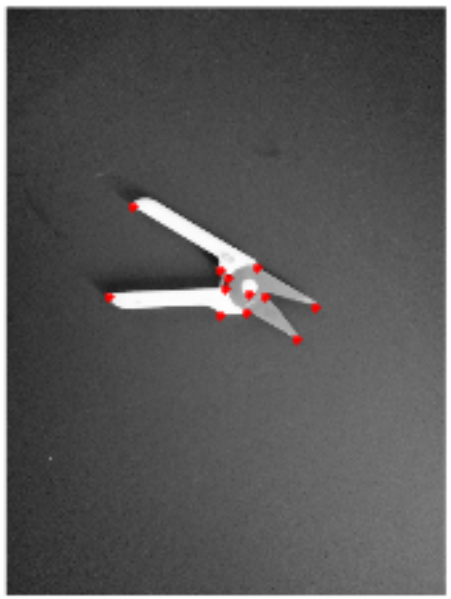


Interest Point Detection for Matching

```
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
```

```
def harrisloc(image,sigma=1.0,area=5.0,hi=0.05,lo=1.0,r=20):
    h = harris(image,sigma=sigma,area=area)
    threshold = max(hi*amax(h),lo*var(h)**.5+mean(h))
    c = AND(h==filters.maximum_filter(h,20),h>threshold)
    l,n = measurements.label(c)
    p = measurements.center_of_mass(c,l,range(1,n))
    return array(p)
```

```
tp = harrisloc(target)
cp = harrisloc(clutter)
subplot(121); axis("off"); imshow(target); plot(tp[:,1],tp[:,0],'r.')
subplot(122); axis("off"); imshow(clutter); plot(cp[:,1],cp[:,0],'r.')
```

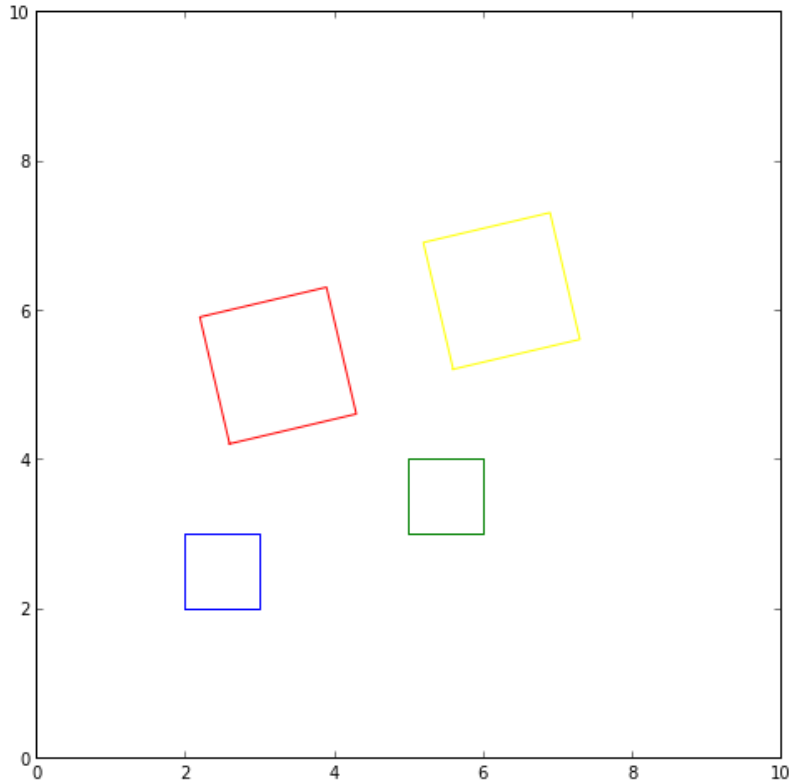


2D Points as Complex Numbers

```
def complex2points(c):
    return c_[real(c),imag(c)]
def points2complex(p):
    return p[:,0]+1j*p[:,1]
```

```
def ppoints(p,*args,**kw):
    if type(p)==complex:
        plot([real(p)],[imag(p)],*args,**kw)
    elif p.ndim==1:
        ps = complex2points(p)
        plot(ps[:,0],ps[:,1],*args,**kw)
    elif p.ndim==2:
        plot(p[:,0],p[:,1],*args,**kw)
    else:
        raise Exception("unknown type")
```

```
M = array([2+2j,3+2j,3+3j,2+3j,2+2j],complex)
translation = 3+1j
rotation = 1.7+0.4j
ppoints(M,color='blue')
ppoints(M+translation,color='green')
ppoints(rotation*points,color='red')
ppoints(rotation*points+translation,color='yellow')
```



```
B = rotation*M+translation
est_rotation = (B[1]-B[0])/(M[1]-M[0])
print rotation,est_rotation
est_translation = B[0]-rotation*M[0]
print translation,est_translation
```

```
(1.7+0.4j) (1.7+0.4j)
(3+1j) (3+1j)
```

```
def evaluate(M,B,translation,rotation,eps=5.0):
    P = rotation*M+translation
    dists = cdist(complex2points(P),complex2points(B))
    min_dists = amin(dists,axis=1)
    return sum(min_dists<eps)
```

```
print evaluate(M,M,0,1,eps=1)
print evaluate(M,rotation*M+translation,0,1,eps=1)
print evaluate(M,rotation*M+translation,translation,rotation,eps=1)
```

```
5
0
5
```

Recognition by Alignment / RANSAC

```
M = points2complex(tp)
B = points2complex(cp)
```

```
best_q = -1
best = None
neval = 0
for i in range(len(M)):
    for j in range(len(M)):
        m1 = M[i]
        m2 = M[j]
        if abs(m1-m2)<10: continue
        for k in range(len(B)):
            for l in range(len(B)):
                b1 = B[k]
                b2 = B[l]
                if abs(b1-b2)<10: continue
                rotscale = (b2-b1)/(m2-m1)
                translation = b1-rotscale*m1
                if abs(rotscale)<0.5 or abs(rotscale)>2.0: continue
                q = evaluate(M,B,translation,rotscale,eps=20)
                neval += 1
                if q>best_q:
                    best_q = q
                    best = (translation,rotscale)
                    print best_q,best
print neval
```

```
5 ((-17.362458453044894+18.323812742944085j), (0.8511633147428313 6+0.4000901357669 9
6 ((-90.387978142076463-74.065573770491824j), (1.1366120218579234+0.4972677595628 4 1
```

```
axis("off"); imshow(clutter.T)
ppoints(B,'go')
translation,rotation = best
ppoints(rotation*M+translation,'r+')
```