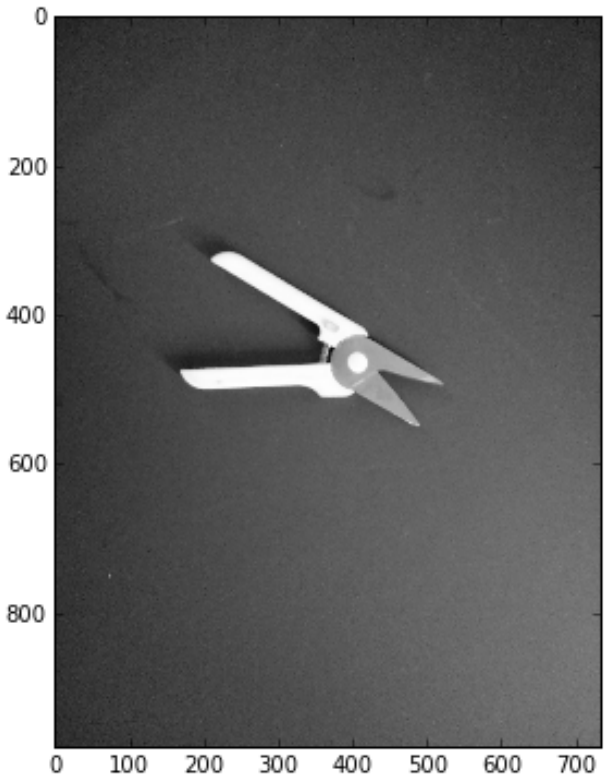
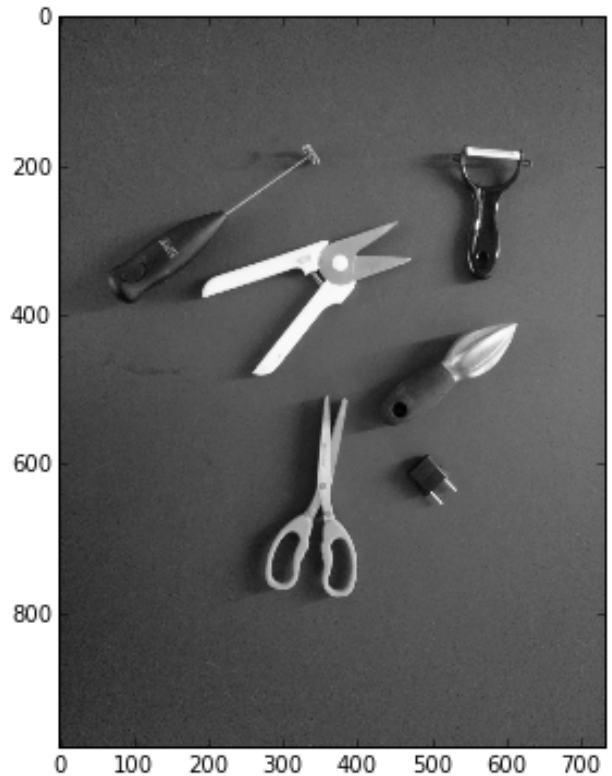
Sample Images

```
target = array(255*interpolation.zoom(mean(imread("match-target.png
    "),axis=2),0.3),'B')
subplot(121); imshow(target)
scene = array(255*interpolation.zoom(mean(imread("match-clutter.png
    "),axis=2),0.3),'B')
subplot(122); imshow(scene)
```

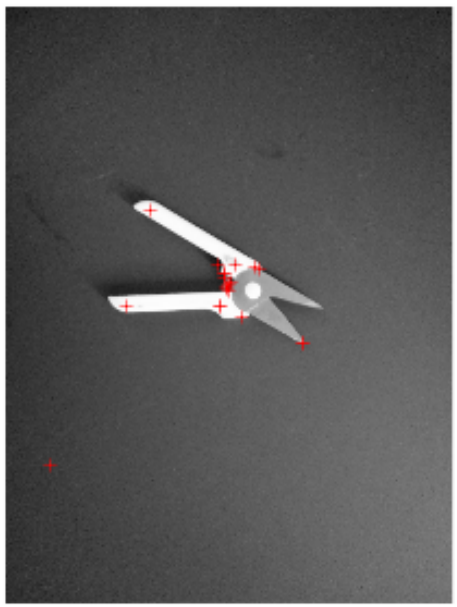


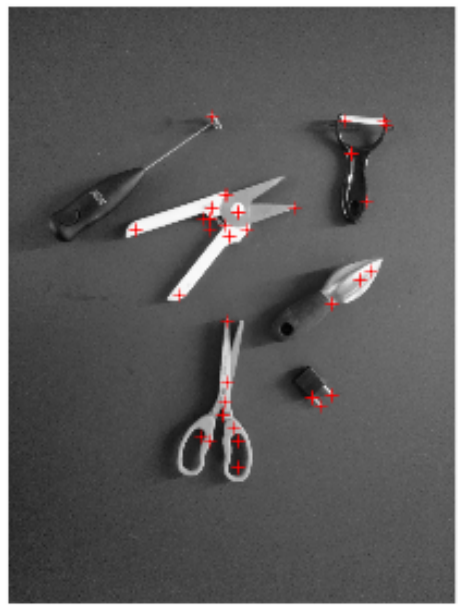


Interest Point Detection for Matching

```
import cv2
sift = cv2.SIFT(20)
tk,td = sift.detectAndCompute(target,None)
sift = cv2.SIFT(50)
sk,sd = sift.detectAndCompute(scene,None)
```

```
p = array([p.pt for p in tk[:50]])
subplot(121); axis("off"); imshow(target); plot(p[:,0],p[:,1],'r+')
p = array([p.pt for p in sk[:50]])
subplot(122); axis("off"); imshow(scene); plot(p[:,0],p[:,1],'r+')
```





Matching the Feature Descriptors

Search through the descriptors in one list of descriptors and find the $k = 2$ best descriptors in the other list.

```
correspondences = cv2.FlannBasedMatcher(dict(algorithm=0,trees=4),
    dict(checks=50))
correspondences = matcher.knnMatch(td,sd,k=2)
print correspondences[0]
```

```
[<DMatch 0x2a73130>, <DMatch 0x2c6adb0>]
```

We find multiple matches per point and disregard matches that are confusingly similar. This restricts the match to descriptors that match fairly unambiguously. This can be justified through a probabilistic log likelihood ratio.

```
1 correspondences = [m for m,n in correspondences if m.distance<0.7*n
      .distance]
2 print len(matches)
3 assert len(matches)>=10
```

12

The `DMatch` objects keep track of the index of the points in the two original descriptor lists.

```
1 print correspondences[7].queryIdx,correspondences[7].trainIdx
```

```
9 21
```

Finding the Geometric Transformation

constructing arrays containing corresponding point coordinates

Construct a list of points to be matched. This might allow multiple matches, but here we just pick the best match for each point.

```
1 source = array([tk[m.queryIdx].pt for m in correspondences],'f').
      reshape(-1,1,2)
2 dest = array([sk[m.trainIdx].pt for m in correspondences],'f').
      reshape(-1,1,2)
```

finding the actual transformation

This uses RANSAC to select a subset of the original points that can be brought into correspondence.

```
1 M,mask = cv2.findHomography(source,dest,cv2.RANSAC,5.0)
2 print M
```

```
[[  4.10828194e-01   1.09055641e+00  -2.22324338e+02]
 [ -9.41693535e-01   9.66230889e-01   3.31551052e+02]
 [ -6.25862802e-04   9.71416522e-04   1.00000000e+00]]
```

computing a boolean mask of matching points

The `mask` result tells us which points actually matched.

```
1 mask = array(mask.ravel())!=0
2 print mask
```

```
[ True  True  True  True  True  True  True False  True  True  True  True]
```

transforming the target points

We transform the matching points.

```
1 tp = array([p.pt for p in tk],'f').reshape(-1,1,2)
2 mp = cv2.perspectiveTransform(tp[mask],M).reshape(-1,2)
```

rendering the transformed target on top of the image

Finally, let's check the match against the image.

```
axis("off"); imshow(scene)
plot(mp[:,0],mp[:,1],'ro')
```