CSE 252A Computer Vision I Fall 2018 - Assignment 3

Instructor: David Kriegman

Assignment Published On: Wednesday, November 7, 2018

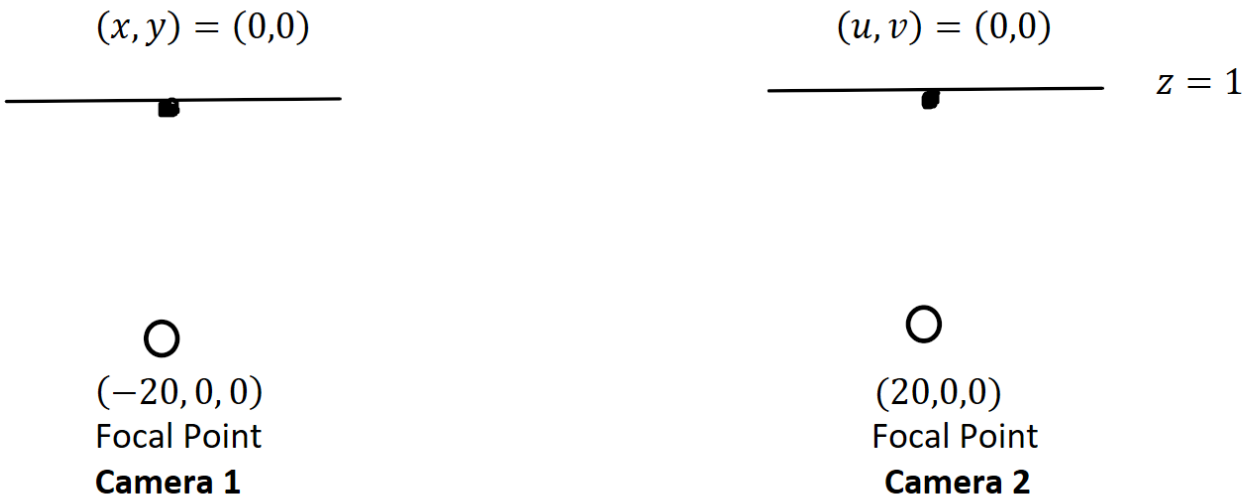
Due On: Tuesday, November 20, 2018 11:59 pm

Instructions

- Review the academic integrity and collaboration policies on the course website.
- This assignment must be completed individually.
- This assignment contains theoretical and programming exercises. If you plan to submit hand written answers for theoretical exercises, please be sure your writing is readable and merge those in order with the final pdf you create out of this notebook. You could fill the answers within the notebook iteself by creating a markdown cell. Please do not mention your explanatory answers in code comments.
- Programming aspects of this assignment must be completed using Python in this notebook.
- If you want to modify the skeleton code, you can do so. This has been provided just to provide you with a framework for the solution.
- You may use python packages for basic linear algebra (you can use numpy or scipy for basic operations), but you may not use packages that directly solve the problem.
- If you are unsure about using a specific package or function, then ask the instructor and teaching assistants for clarification.
- You must submit this notebook exported as a pdf. You must also submit this notebook as .ipynb file.
- You must submit both files (.pdf and .ipynb) on Gradescope. You must mark each problem on Gradescope in the pdf.
- **Late policy** - 10% per day late penalty after due date up to 3 days.

Problem 1: Epipolar Geometry [3 pts]

Consider two cameras whose image planes are the z=1 plane, and whose focal points are at (-20, 0, 0) and (20, 0, 0). We''ll call a point in the first camera (x, y), and a point in the second camera (u, v). Points in each camera are relative to the camera center. So, for example if (x, y) = (0, 0), this is really the point (-20, 0, 1) in world coordinates, while if (u, v) = (0, 0) this is the point (20, 0, 1).



a) Suppose the points (x, y) = (12, 12) is matched to the point (u, v) = (1, 12). What is the 3D location of this point?

b) Consider points that lie on the line x + z = 0, y = 0. Use the same stereo set up as before. Write an analytic expression giving the disparity of a point on this line after it projects onto the two images, as a function of its position in the right image. So your expression should only involve the variables u and d (for disparity). Your expression only needs to be valid for points on the line that are in front of the cameras, i.e. with z > 1.

CSE 252A HW#3 Zhisheng Huang A53240987

P1/

(a) Since $(x, y) = (0, 0) \rightarrow (-20, 0, 1)$
$(u, v) = (0, 0) \rightarrow (20, 0, 1)$,
$(x, y) = (12, 12) \rightarrow (-8, 12, 1)$
$(u, v) = (1, 12) \rightarrow (21, 12, 1)$

For points in each camera are relative to the camera center, we can get the expressions for two lines which go through these two points.

$L_1 = (-20, 0, 0) + a((-8, 12, 1) - (-20, 0, 0))$
$L_2 = (20, 0, 0) + b((21, 12, 1) - (20, 0, 0))$

Let $L_1 = L_2$

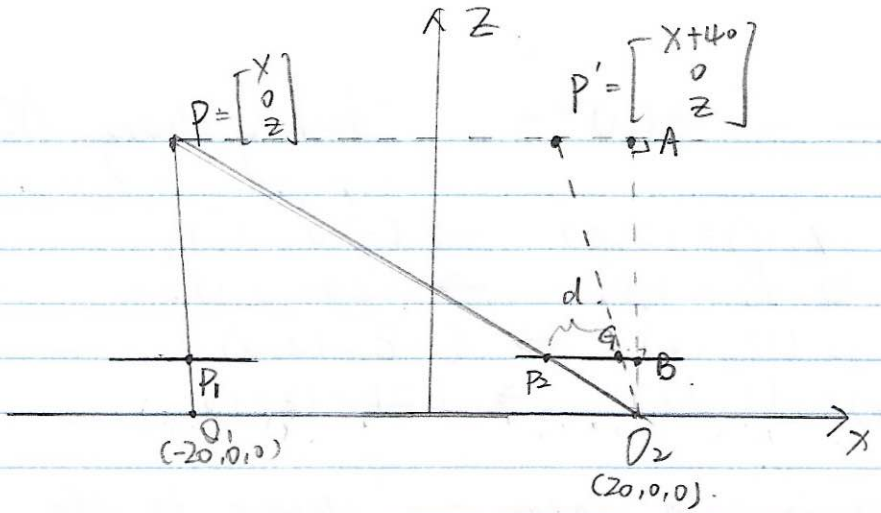
We get $\begin{pmatrix} -20 + 12a \\ 12a \\ a \end{pmatrix} = \begin{pmatrix} 20 + b \\ 12b \\ b \end{pmatrix}$

So, $\begin{cases} a = b \\ -20 + 12a = 20 + b \end{cases} \Rightarrow a = b = \frac{40}{11}$

Thus, the 3D location of the point is $\begin{bmatrix} \frac{200}{11} \\ \frac{480}{11} \\ \frac{40}{11} \end{bmatrix}$

(b) Let's assume P's on this line, with coordinates as $\begin{bmatrix} x \\ 0 \\ z \end{bmatrix}$, where $x + z = 0$.
P' is the corresponding point with coordinate $\begin{bmatrix} x+40 \\ 0 \\ z \end{bmatrix}$.

Two points on 2 images are P_1 & P_2. G is the corresponding point for P_1 on image II, which remains the same distance to the center of image II.
The illustration graph is shown below



From the graph, we can tell.

$AO_2 \perp P_2B$,

$A = \begin{bmatrix} 20 \\ 0 \\ z \end{bmatrix}$, $B = \begin{bmatrix} 20 \\ 0 \\ 1 \end{bmatrix}$

we want to get the expression for $d = GP_2$

$$\frac{d}{PP'} = \frac{O_2G}{O_2P'} = \frac{O_2B}{O_2A} = \frac{P_2B}{PA}$$

$$= \frac{-x}{20-x} = \frac{1}{z} = \frac{1}{-x}$$

So, $d = -2x - 2$

Problem 2: Epipolar Rectification [4 pts]

In stereo vision, image rectification is a common preprocessing step to simplify the problem of finding matching points between images. The goal is to warp image views such that the epipolar lines are horizontal scan lines of the input images. Suppose that we have captured two images I_A and I_B from identical calibrated cameras separated by a rigid transformation

$$\begin{matrix}B\\A\end{matrix}\mathrm{T} = \begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix}$$

Without loss of generality assume that camera A's optical center is positioned at the origin and that its optical axis is in the direction of the z-axis.

From the lecture, a rectifying transform for each image should map the epipole to a point infinitely far away in the horizontal direction $H_A e_A = H_B e_B = [1, 0, 0]^T$. Consider the following special cases:

a) Pure horizontal translation $t = [t_x, 0, 0]^T$, R = I

b) Pure translation orthogonal to the optical axis $t = [t_x, t_y, 0]^T$, R = I

c) Pure translation along the optical axis $t = [0, 0, t_z]^T$, R = I

d) Pure rotation $t = [0, 0, 0]^T$, R is an arbitrary rotation matrix

For each of these cases, determine whether or not epipolar rectification is possible. Include the following information for each case

- The epipoles e_A and e_B
- The equation of the epipolar line l_B in I_B corresponding to the point $[x_A, y_A, 1]^T$ in I_A (if one exists)
- A plausible solution to the rectifying transforms H_A and H_B (if one exists) that attempts to minimize distortion (is as close as possible to a 2D rigid transformation). Note that the above 4 cases are special cases; a simple solution should become apparent by looking at the epipolar lines.

One or more of the above rigid transformations may be a degenerate case where rectification is not possible or epipolar geometry does not apply. If so, explain why.

Problem 3: Filtering [3 pts]

a) Consider smoothing an image with a 3x3 box filter and then computing the derivative in the x direction. What is a single convolution kernel that will implement this operation?

b) Give an example of a separable filter and compare the number of arithmetic operations it takes to convolve using that filter on an $n \times n$ image before and after separation.

Problem 4: Sparse Stereo Matching [22 pts]

P2/ From the problem, we know that

$$^{B}_{A}T = \begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix}$$

a) Pure horizontal translation. $t = [t_x, 0, 0]^T$, $R = I$

$$E = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -t_x \\ 0 & t_x & 0 \end{bmatrix} = [t_x]R$$

1) from the lecture, we know that $\begin{cases} E e_B = 0 \\ E^T e_A = 0 \end{cases}$

$\Rightarrow$, we get $e_A = e_B = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$

2) $\begin{bmatrix} a \\ b \\ c \end{bmatrix} = E^T \begin{bmatrix} x_A \\ y_A \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & t_x \\ 0 & -t_x & 0 \end{bmatrix} \begin{bmatrix} x_A \\ y_A \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ t_x \\ -t_x y_A \end{bmatrix}$

So, l_B: $t_x \cdot y - t_x y_A = 0 \Rightarrow l_B$: $y = y_A$.

3) assume $e_1 = e_A = [1, 0, 0]$

$$e_2 = \frac{1}{\sqrt{e_{1x}^2 + e_{1y}^2}} [-e_{1y}, e_{1x}, 0]^T = [0, 1, 0]^T$$

$e_3 = e_1 \times e_3$

$= [0, 0, 1]^T$

$$\therefore H_A = \begin{bmatrix} e_1^T \\ e_2^T \\ e_3^T \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad H_B = RHA = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Thus, epipolar rectification is possible

b) Given $t = [t_x, t_y, 0]$, $R = I$.

$$E = \begin{bmatrix} 0 & 0 & t_y \\ 0 & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix}$$

1) According to $\begin{cases} E e_B = 0 \\ E^T e_A = 0 \end{cases}$

$$\Rightarrow e_A = e_B = \left[\frac{t_x}{\sqrt{t_x^2 + t_y^2}}, \frac{t_y}{\sqrt{t_x^2 + t_y^2}}, 0 \right]^T$$

2) $\begin{bmatrix} a \\ b \\ c \end{bmatrix} = E^T \begin{bmatrix} x_A \\ y_A \\ 1 \end{bmatrix} = \begin{bmatrix} -t_y \\ t_x \\ t_y x_A - t_x y_A \end{bmatrix}$

$\therefore L_B = -t_y x + t_y y + t_y x_A - t_x y_A = 0$

3) $e_1 = e_A$, $e_2 = \frac{1}{\sqrt{e_{1x}^2 + e_{1y}^2}} [-e_{1y}, e_{1x}, 0]^T$

$= \left[\frac{-t_y}{\sqrt{t_x^2 + t_y^2}}, \frac{t_x}{\sqrt{t_x^2 + t_y^2}}, 0 \right]^T$

$e_3 = e_1 \times e_2 = [0, 0, 1]^T$

$H_A = \begin{bmatrix} e_1^T \\ e_2^T \\ e_3^T \end{bmatrix} = \begin{bmatrix} \frac{t_x}{\sqrt{t_x^2 + t_y^2}} & \frac{t_y}{\sqrt{t_x^2 + t_y^2}} & 0 \\ \frac{-t_y}{\sqrt{t_x^2 + t_y^2}} & \frac{t_x}{\sqrt{t_x^2 + t_y^2}} & 0 \\ 0 & 0 & 1 \end{bmatrix} = R H_A = H_B.$

Thus, epipolar rectification is possible.

c) Given $t = [0, 0, t_z]^T$, $R = I$

$E = \begin{bmatrix} 0 & -t_z & 0 \\ t_z & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$

1) From $\begin{cases} E e_B = 0 \\ E^T e_A = 0 \end{cases} \Rightarrow e_A = e_B = [0, 0, 1]^T$

2) $\begin{bmatrix} a \\ b \\ c \end{bmatrix} = E^T \begin{bmatrix} x_A \\ y_A \\ 1 \end{bmatrix} = \begin{bmatrix} t_z y_A \\ -t_z x_A \\ 0 \end{bmatrix}$

$\therefore L_B : t_z y_A x - t_z x_A y = 0 \Rightarrow L_B : y_A x - x_A y = 0$

3) $e_1 = e_A = [0, 0, 1]^T$.

$e_2 = [0, 1, 0]^T$

$e_3 = e_1 \times e_2 = [1, 0, 0]^T$

$$H_A = \begin{bmatrix} e_1^T \\ e_2^T \\ e_3^T \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

$$H_B = RH_A = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

Thus, Epipolar rectification's possible

d). Given $t = [0, 0, 0]^T$, R's an arbitary rotation matrix.

$E = [t_\times] R$

1) $\begin{cases} E e_B = 0 \\ E^T e_A = 0 \end{cases} \Rightarrow$ we get e_A & e_B are arbitary unit vectors

2) $\begin{bmatrix} a \\ b \\ c \end{bmatrix} = E^T \begin{bmatrix} x_A \\ y_A \\ 1 \end{bmatrix} = 0$

NO such line exists.

3) NO such H_A & H_B exist

Thus, epipolar rectification's impossible.

P3/ Filtering

a)

$$\frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix} .* \begin{bmatrix}1&0&-1\\2&0&-2\\1&0&-1\end{bmatrix}$$

$$= \frac{1}{9}\begin{bmatrix}1&0&-1\\2&0&-2\\1&0&-1\end{bmatrix}$$

b) Assume a separable filter is $M \in R^{m \times m}$.

$$M = a b^T \quad \text{where } a \in R^m,\ b \in R^m$$

Before separation, we use M to convolve, $(n-m+1)^2$ arithmetic operations.

After separation, we use a, b^T to convolve, $(n-m+1) \times n \times 2 = 2n(n-m+1)$.

For example,

$$M = \frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix} = \frac{1}{3}\begin{bmatrix}1\\1\\1\end{bmatrix} * \frac{1}{3}\begin{bmatrix}1&1&1\end{bmatrix}$$

For an $n \times n$ image, without padding

$(n-2)^2$ operations for using M before separation

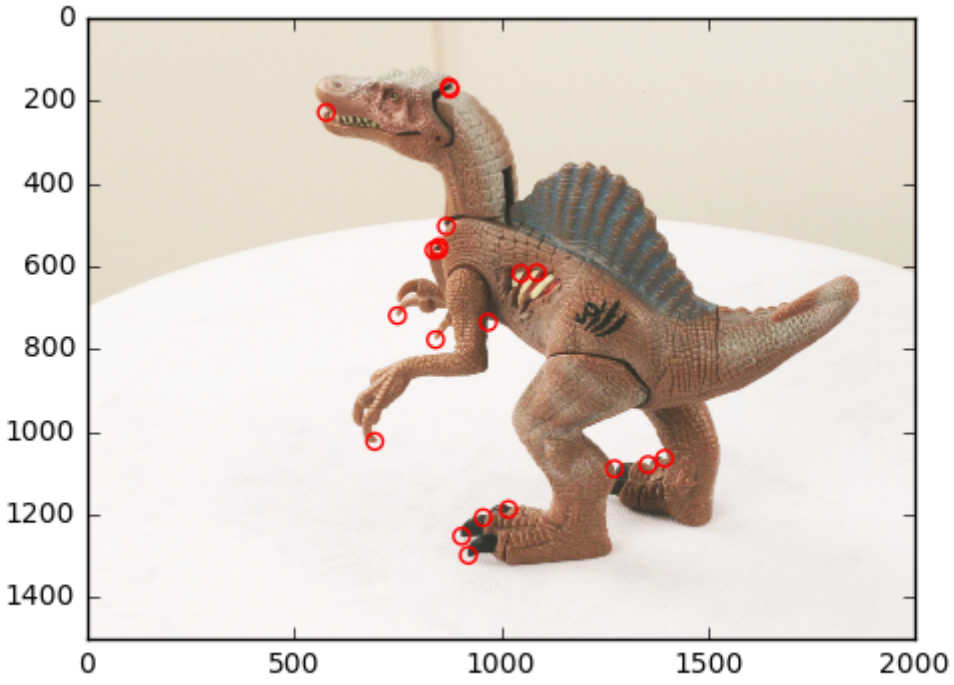
$2(n-2)n$ operations after separation

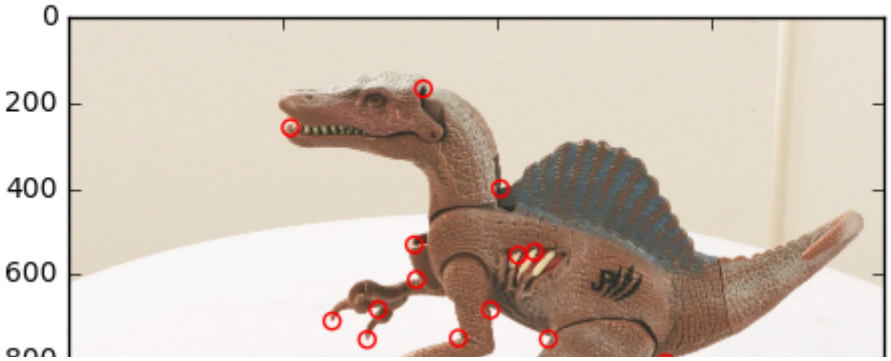
In this problem we will play around with sparse stereo matching methods. You will work on two image pairs, a warrior figure and a figure from the Matrix movies. These files both contain two images, two camera matrices, and set sets of corresponding points (extracted by manually clicking the images). For illustration, I have run my code on a third image pair (dino1.png, dino2.png). This data is also provided for you to debug your code, but you should only report results on warrior and matrix. In other words, where I include one (or a pair) of images in the assignment below, you will provide the same thing but for BOTH matrix and warrior. Note that the matrix image pair is harder, in the sense that the matching algorithms we are implementing will not work quite as well. You should expect good results, however, on warrior.

Corner Detection [5 pts]

The first thing we need to do is to build a corner detector. This should be done according to http://cseweb.ucsd.edu/classes/fa18/cse252A-a/lec11.pdf (http://cseweb.ucsd.edu/classes/fa18/cse252A-a/lec11.pdf). You should fill in the function `corner_detect` below, and take as `input` `corner_detect(image, nCorners, smoothSTD, windowSize)` where smoothSTD is the standard deviation of the smoothing kernel and windowSize is the window size for corner detector and non maximum suppression. In the lecture the corner detector was implemented using a hard threshold. Do not do that but instead return the nCorners strongest corners after non-maximum suppression. This way you can control exactly how many corners are returned. Run your code on all four images (with nCorners = 20) and show outputs as in Figure 2. You may find scipy.ndimage.filters.gaussian_filter easy to use for smoothing. In this problem, try different parameters and then comment on results.

1. windowSize = 3, 5, 9, 17
2. smoothSTD = 0.5, 1, 2, 4





In [195]:
```
import numpy as np
from scipy.misc import imread
import matplotlib.pyplot as plt
from scipy.ndimage.filters import gaussian_filter
```

In [196]:
```
def rgb2gray(rgb):
    """ Convert rgb image to grayscale.
    """
    return np.dot(rgb[...,:3], [0.299, 0.587, 0.114])
```

In [199]:
```
def corner_detect(image, nCorners, smoothSTD, windowSize):
    """Detect corners on a given image.

    Args:
        image: Given a grayscale image on which to detect corners.
        nCorners: Total number of corners to be extracted.
        smoothSTD: Standard deviation of the Gaussian smoothing kernel.
        windowSize: Window size for corner detector and non maximum supp
ression.

    Returns:
        Detected corners (in image coordinate) in a numpy array (n*2).

    """

    """
    Your code here:
    """
    corners = np.zeros((nCorners, 2))
    ImageSmooth = gaussian_filter(image, sigma=smoothSTD)
    Img_dy, Img_dx = np.gradient(ImageSmooth)
    window = np.ones([windowSize, windowSize])

    Ix_2 = findC(Img_dx * Img_dx, window)
    Iy_2 = findC(Img_dy * Img_dy, window)
    Ixy_2 = findC(Img_dx * Img_dy, window)

    e = np.zeros(image.shape)  ## find the minimum eigen values
    for i in xrange(image.shape[0]):
        for j in xrange(image.shape[1]):
            c = np.array([[Ix_2[i][j], Ixy_2[i][j]],[Ixy_2[i][j], Iy_2[i
][j]]])
            lamda,_ = np.linalg.eig(c)
            e[i][j] = np.min((lamda[0], lamda[1])) ## store the minimum
 eigen values of matric C for the entire image

    localMax = findMaxLocal(window, e) # return (i, j, eigenvalue)
    aaa = np.unique(localMax[:, 2])
    th = aaa[-nCorners]
    ind = np.where(localMax[:,2] > th)
    corners = localMax[ind][:,:2]

    return corners


def findC(img, window):

    winRowHalf = int((window.shape[0]- 1) / 2)
    winColHalf = int((window.shape[1] - 1) / 2)
    Row = img.shape[0]
    Col = img.shape[1]
    res = np.zeros((Row, Col))
    for i in xrange(winRowHalf, Row - winRowHalf):
        for j in xrange(winColHalf, Col - winColHalf):
            res[i][j] = (img[i - winRowHalf : i + winRowHalf+1, j - winR
owHalf : j + winColHalf+1] * window).sum()
```

```
    return res

def findMaxLocal(window, eigenM):
    winRowHalf = int((window.shape[0]- 1) / 2)
    winColHalf = int((window.shape[1] - 1) / 2)
    Row = eigenM.shape[0]
    Col = eigenM.shape[1]
    res = np.zeros((Row, Col))
    temp = np.zeros(window.shape)
    corners = np.zeros((Row * Col, 3))
    counter = 1
    for i in xrange(winRowHalf, Row - winRowHalf):
        for j in xrange(winColHalf, Col - winColHalf):
            temp = (eigenM[i - winRowHalf : i + winRowHalf+1, j - winRow
Half : j + winColHalf+1])
            if temp[winRowHalf][winColHalf]  == temp.max():
                #res[i][j] = True
                #corners = np.vstack((corners, np.array((j, i, eigenM[i]
[j]))))
                corners[counter,:] = np.array((j, i, eigenM[i][j]))
                counter += 1

    return corners
```

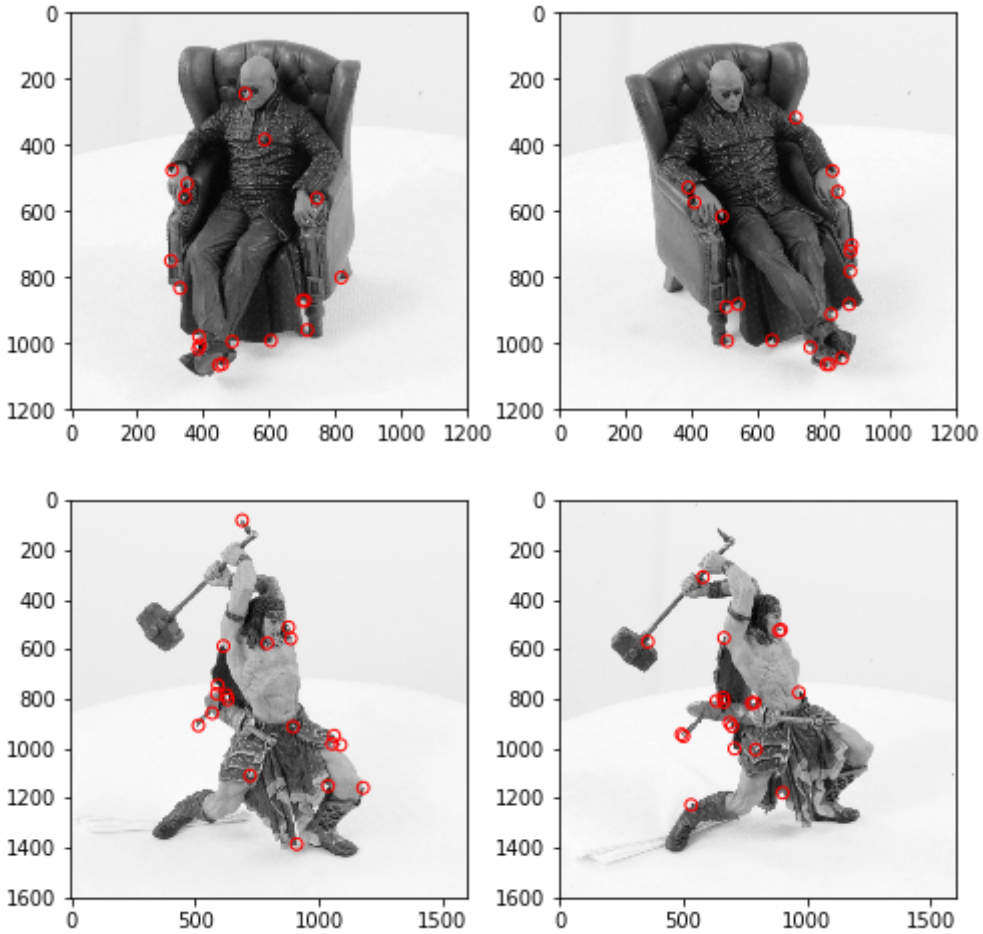
In [201]:

```
img = np.ones((15,15))
window = np.ones((5,5))
res = findC(img, window)
# print(res.max())
eigenM = np.vstack((np.ones((10, 15)) / 25.0, np.ones((5, 15)) / 1.0))
aaa = findMaxLocal(window, eigenM)
# print(aaa)
np.unique(aaa[:,2])
np.min((888, 2999))
c = np.array([[1,2], [5,6]])
w, v = np.linalg.eig(c)
ddd = np.where(eigenM > 0.5)



test = np.vstack((np.ones((10,3))/7.0, np.ones((10,3))/2.0))
th = 0.3
testres = np.where(test[:,2] > th)
# print("test: ", test)
# print("testres: ", testres)
#test[testres][:, :2]
```

In [203]:
```
# detect corners on warrior and matrix sets
# adjust your corner detection parameters here
nCorners = 20
smoothSTD = 2
windowSize = 11

# read images and detect corners on images
imgs_mat = []
crns_mat = []
imgs_war = []
crns_war = []
for i in range(2):
    img_mat = imread('p4/matrix/matrix' + str(i) + '.png')
    imgs_mat.append(rgb2gray(img_mat))
    # downsize your image in case corner_detect runs slow in test
    # imgs_mat.append(rgb2gray(img_mat)[::2, ::2])
    crns_mat.append(corner_detect(imgs_mat[i], nCorners, smoothSTD, wind
owSize))

    img_war = imread('p4/warrior/warrior' + str(i) + '.png')
    imgs_war.append(rgb2gray(img_war))
    # downsize your image in case corner_detect runs slow in test
    # imgs_war.append(rgb2gray(img_war)[::2, ::2])
    crns_war.append(corner_detect(imgs_war[i], nCorners, smoothSTD, wind
owSize))
```

```
/Users/huangzhisheng/anaconda2/lib/python2.7/site-packages/ipykernel_la
uncher.py:13: DeprecationWarning: `imread` is deprecated!
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.
  del sys.path[0]
/Users/huangzhisheng/anaconda2/lib/python2.7/site-packages/ipykernel_la
uncher.py:19: DeprecationWarning: `imread` is deprecated!
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.
```

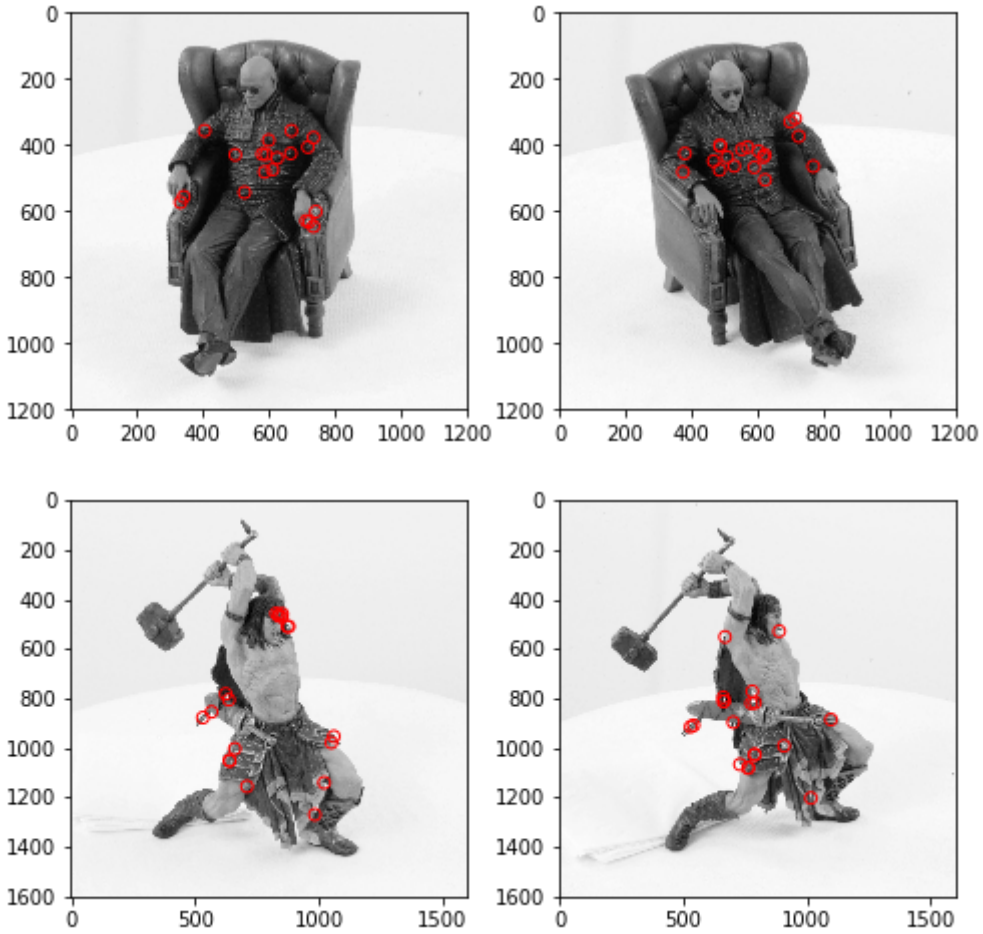
In [206]:
```
def show_corners_result(imgs, corners):
    fig = plt.figure(figsize=(8, 8))
    ax1 = fig.add_subplot(221)
    ax1.imshow(imgs[0], cmap='gray')
    ax1.scatter(corners[0][:, 0], corners[0][:, 1], s=35, edgecolors='r'
, facecolors='none')

    ax2 = fig.add_subplot(222)
    ax2.imshow(imgs[1], cmap='gray')
    ax2.scatter(corners[1][:, 0], corners[1][:, 1], s=35, edgecolors='r'
, facecolors='none')
    plt.show()

show_corners_result(imgs_mat, crns_mat)
show_corners_result(imgs_war, crns_war)
#print(crns_mat)
```



In [40]:
```
# detect corners on warrior and matrix sets
# adjust your corner detection parameters here
nCorners = 20
smoothSTD = .5
windowSize = 3

# read images and detect corners on images
imgs_mat = []
crns_mat = []
imgs_war = []
crns_war = []
for i in range(2):
    img_mat = imread('p4/matrix/matrix' + str(i) + '.png')
    imgs_mat.append(rgb2gray(img_mat))
    # downsize your image in case corner_detect runs slow in test
    # imgs_mat.append(rgb2gray(img_mat)[::2, ::2])
    crns_mat.append(corner_detect(imgs_mat[i], nCorners, smoothSTD, wind
owSize))

    img_war = imread('p4/warrior/warrior' + str(i) + '.png')
    imgs_war.append(rgb2gray(img_war))
    # downsize your image in case corner_detect runs slow in test
    # imgs_war.append(rgb2gray(img_war)[::2, ::2])
    crns_war.append(corner_detect(imgs_war[i], nCorners, smoothSTD, wind
owSize))



def show_corners_result(imgs, corners):
    fig = plt.figure(figsize=(8, 8))
    ax1 = fig.add_subplot(221)
    ax1.imshow(imgs[0], cmap='gray')
    ax1.scatter(corners[0][:, 0], corners[0][:, 1], s=35, edgecolors='r'
, facecolors='none')

    ax2 = fig.add_subplot(222)
    ax2.imshow(imgs[1], cmap='gray')
    ax2.scatter(corners[1][:, 0], corners[1][:, 1], s=35, edgecolors='r'
, facecolors='none')
    plt.show()

print("smoothSTD = ", smoothSTD)
print("window size = ", windowSize)
show_corners_result(imgs_mat, crns_mat)
show_corners_result(imgs_war, crns_war)
```

```
/Users/huangzhisheng/anaconda2/lib/python2.7/site-packages/ipykernel_la
uncher.py:13: DeprecationWarning: `imread` is deprecated!
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.
  del sys.path[0]
/Users/huangzhisheng/anaconda2/lib/python2.7/site-packages/ipykernel_la
uncher.py:19: DeprecationWarning: `imread` is deprecated!
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.
```

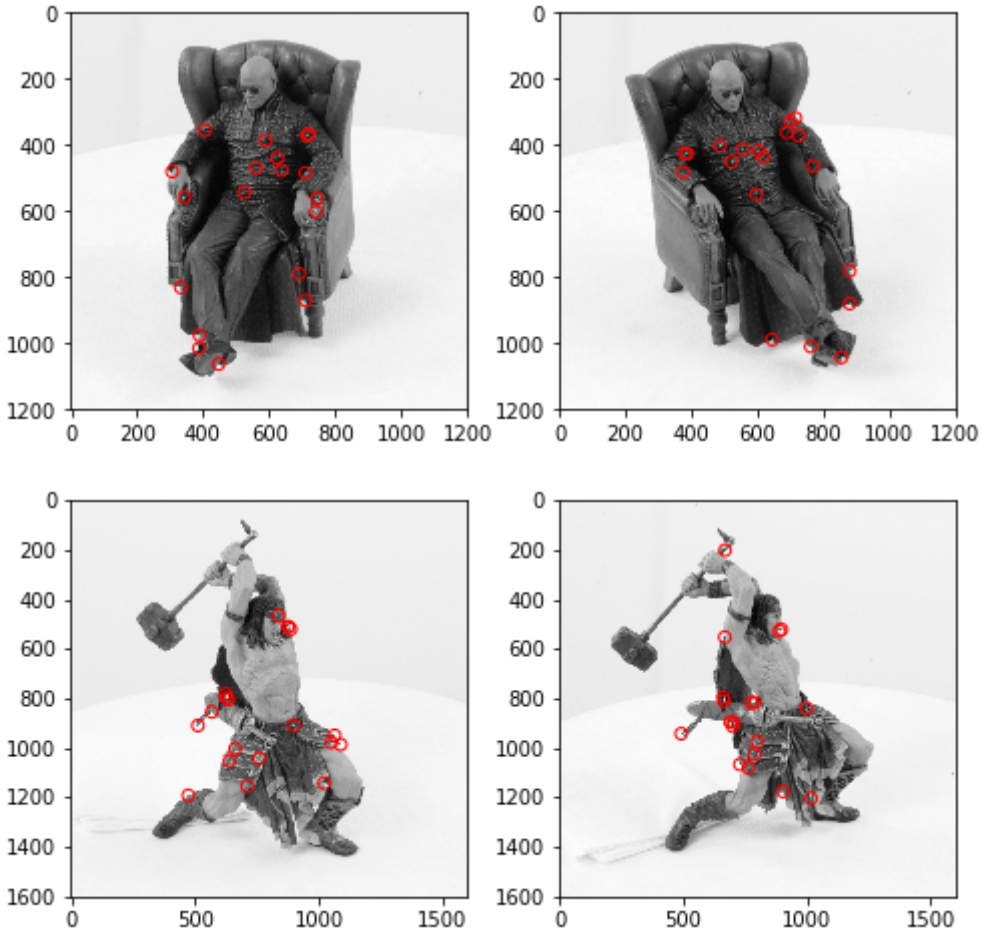
```
('smoothSTD = ', 0.5)
('window size = ', 3)
```



In [41]:
```
# detect corners on warrior and matrix sets
# adjust your corner detection parameters here
nCorners = 20
smoothSTD = 1
windowSize = 5

# read images and detect corners on images
imgs_mat = []
crns_mat = []
imgs_war = []
crns_war = []
for i in range(2):
    img_mat = imread('p4/matrix/matrix' + str(i) + '.png')
    imgs_mat.append(rgb2gray(img_mat))
    # downsize your image in case corner_detect runs slow in test
    # imgs_mat.append(rgb2gray(img_mat)[::2, ::2])
    crns_mat.append(corner_detect(imgs_mat[i], nCorners, smoothSTD, wind
owSize))

    img_war = imread('p4/warrior/warrior' + str(i) + '.png')
    imgs_war.append(rgb2gray(img_war))
    # downsize your image in case corner_detect runs slow in test
    # imgs_war.append(rgb2gray(img_war)[::2, ::2])
    crns_war.append(corner_detect(imgs_war[i], nCorners, smoothSTD, wind
owSize))



def show_corners_result(imgs, corners):
    fig = plt.figure(figsize=(8, 8))
    ax1 = fig.add_subplot(221)
    ax1.imshow(imgs[0], cmap='gray')
    ax1.scatter(corners[0][:, 0], corners[0][:, 1], s=35, edgecolors='r'
, facecolors='none')

    ax2 = fig.add_subplot(222)
    ax2.imshow(imgs[1], cmap='gray')
    ax2.scatter(corners[1][:, 0], corners[1][:, 1], s=35, edgecolors='r'
, facecolors='none')
    plt.show()

print("smoothSTD = ", smoothSTD)
print("window size = ", windowSize)
show_corners_result(imgs_mat, crns_mat)
show_corners_result(imgs_war, crns_war)
```

```
/Users/huangzhisheng/anaconda2/lib/python2.7/site-packages/ipykernel_la
uncher.py:13: DeprecationWarning: `imread` is deprecated!
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.
  del sys.path[0]
/Users/huangzhisheng/anaconda2/lib/python2.7/site-packages/ipykernel_la
uncher.py:19: DeprecationWarning: `imread` is deprecated!
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.
```

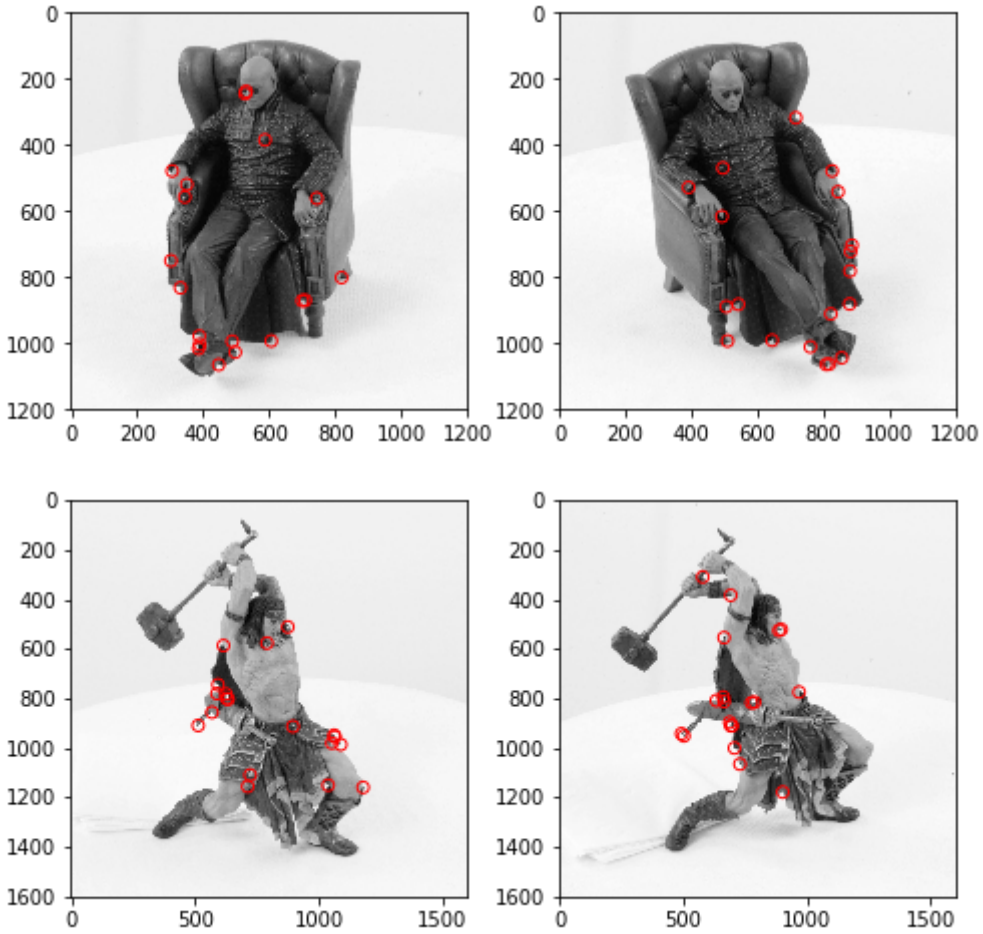
```
('smoothSTD = ', 1)
('window size = ', 5)
```



In [42]:
```
# detect corners on warrior and matrix sets
# adjust your corner detection parameters here
nCorners = 20
smoothSTD = 2
windowSize = 9

# read images and detect corners on images
imgs_mat = []
crns_mat = []
imgs_war = []
crns_war = []
for i in range(2):
    img_mat = imread('p4/matrix/matrix' + str(i) + '.png')
    imgs_mat.append(rgb2gray(img_mat))
    # downsize your image in case corner_detect runs slow in test
    # imgs_mat.append(rgb2gray(img_mat)[::2, ::2])
    crns_mat.append(corner_detect(imgs_mat[i], nCorners, smoothSTD, wind
owSize))

    img_war = imread('p4/warrior/warrior' + str(i) + '.png')
    imgs_war.append(rgb2gray(img_war))
    # downsize your image in case corner_detect runs slow in test
    # imgs_war.append(rgb2gray(img_war)[::2, ::2])
    crns_war.append(corner_detect(imgs_war[i], nCorners, smoothSTD, wind
owSize))



def show_corners_result(imgs, corners):
    fig = plt.figure(figsize=(8, 8))
    ax1 = fig.add_subplot(221)
    ax1.imshow(imgs[0], cmap='gray')
    ax1.scatter(corners[0][:, 0], corners[0][:, 1], s=35, edgecolors='r'
, facecolors='none')

    ax2 = fig.add_subplot(222)
    ax2.imshow(imgs[1], cmap='gray')
    ax2.scatter(corners[1][:, 0], corners[1][:, 1], s=35, edgecolors='r'
, facecolors='none')
    plt.show()

print("smoothSTD = ", smoothSTD)
print("window size = ", windowSize)
show_corners_result(imgs_mat, crns_mat)
show_corners_result(imgs_war, crns_war)
```

```
/Users/huangzhisheng/anaconda2/lib/python2.7/site-packages/ipykernel_la
uncher.py:13: DeprecationWarning: `imread` is deprecated!
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.
  del sys.path[0]
/Users/huangzhisheng/anaconda2/lib/python2.7/site-packages/ipykernel_la
uncher.py:19: DeprecationWarning: `imread` is deprecated!
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.
```

```
('smoothSTD = ', 2)
('window size = ', 9)
```



Comments

According to the experiment results, the window size has a large effect on the corner detection. If we use larger window size, we need to wait for a long time to get the results and will get better accuracy. The detected corners are separate and locations are more appropriate. But if the window size is too small, the feature cannot be detected by the window very well. Some corners may be very close. The gaussina filter is used to reduce the noise of the image. If the smoothSTD is very large, the image will be more smooth since the noise of the image has been largely romoved, which makes it more difficult to detect the edge. But small smoothSTDs always bring too many detected corners. So, we can avoid detecting the inappropriate corners by using appropriate smoothSTDs.

NCC (Normalized Cross-Correlation) Matching [2 pts]

Write a function nccmatch that implements the NCC matching algorithm for two input windows. NCC = $\sum{i,j}\tilde{W_1} (i,j)\cdot \tilde{W2} (i,j)where\tilde{W} = \frac{W - \overline{W}}{\sqrt{\sum{k,l}(W(k,l) - \overline{W})^2}}isamean − shiftedandnormalizedversionofthewindowand\overline{W}$ is the mean pixel value in the window W.

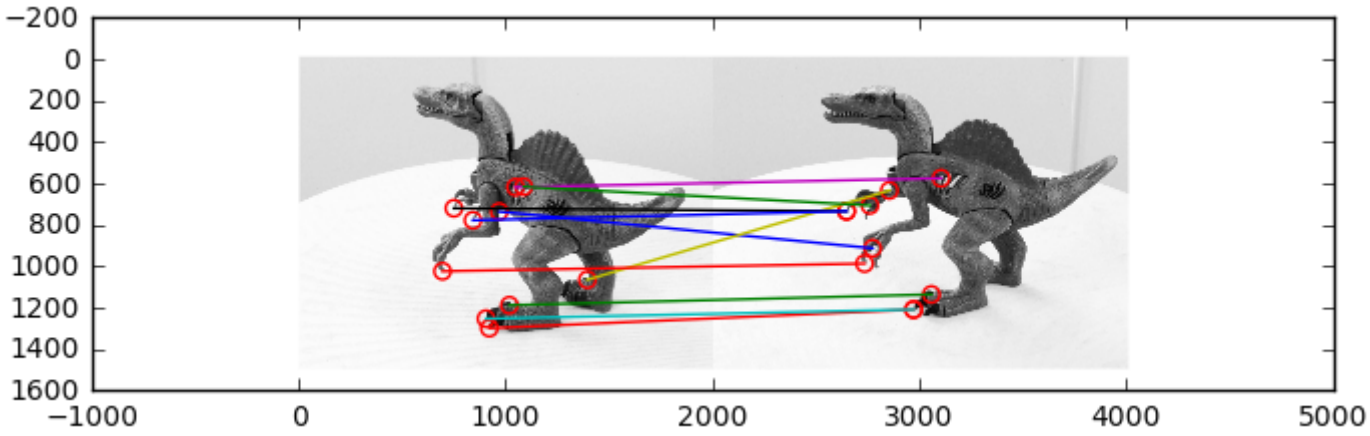
In [309]:
```
def ncc_match(img1, img2, c1, c2, R):
    """Compute NCC given two windows.

    Args:
        img1: Image 1.
        img2: Image 2.
        c1: Center (in image coordinate) of the window in image 1.
        c2: Center (in image coordinate) of the window in image 2.
        R: R is the radius of the patch, 2 * R + 1 is the window size

    Returns:
        NCC matching score for two input windows.

    """

    """
    Your code here:
    """
    matching_score = 0

    w1 = img1[int(c1[1] - R) : int(c1[1] + R + 1), int(c1[0] - R):int(c1
[0] + R+1)] ## for image [c1[1], c1[0]]
    w2 = img2[int(c2[1] - R) : int(c2[1] + R + 1), int(c2[0] - R):int(c2
[0] + R+1)]
    #print(int(c1[0] - R),int(c1[0] + R+1))
    #print(c1)
    w1_mean = w1.mean()
    w2_mean = w2.mean()
    w1_ = (w1 - w1_mean) / np.sqrt(((w1 - w1_mean)**2).sum())
    w2_ = (w2 - w2_mean) / np.sqrt(((w2 - w2_mean)**2).sum())
    matching_score = (w1_ * w2_).sum()

    return matching_score
```

In [310]:
```
# test NCC match
img1 = np.array([[1, 2, 3, 4], [4, 5, 6, 8], [7, 8, 9, 4]])
img2 = np.array([[1, 2, 1, 3], [6, 5, 4, 4], [9, 8, 7, 3]])
print ncc_match(img1, img2, np.array([1, 1]), np.array([1, 1]), 1)
# should print 0.8546
print ncc_match(img1, img2, np.array([2, 1]), np.array([2, 1]), 1)
# should print 0.8457
print ncc_match(img1, img2, np.array([1, 1]), np.array([2, 1]), 1)
# should print 0.6258
```

```
0.8546547739343037
0.8457615282174419
0.6258689611426174
```

Naive Matching [4 pts]

Equipped with the corner detector and the NCC matching function, we are ready to start finding correspondances. One naive strategy is to try and find the best match between the two sets of corner points. Write a script that does this, namely, for each corner in image1, find the best match from the detected corners in image2 (or, if the NCC match score is too low, then return no match for that point). You will have to figure out a good threshold (NCCth) value by experimentation. Write a function naiveCorrespondanceMatching.m and call it as below. Examine your results for 10, 20, and 30 detected corners in each image. Choose a number of detected corners to the maximize the number of correct matching pairs. naive_matching will call your NCC mathching code.



In [226]:
```
def naive_matching(img1, img2, corners1, corners2, R, NCCth):
    """Compute NCC given two windows.

    Args:
        img1: Image 1.
        img2: Image 2.
        corners1: Corners in image 1 (nx2)
        corners2: Corners in image 2 (nx2)
        R: NCC matching radius
        NCCth: NCC matching score threshold

    Returns:
        NCC matching result a list of tuple (c1, c2),
        c1 is the 1x2 corner location in image 1,
        c2 is the 1x2 corner location in image 2.

    """

    """
    Your code here:
    """
    matching = []
    best_match = np.ones(corners1.shape[0]) # store the best match
    best_match = best_match * NCCth # NCC threshold
    for i in xrange(corners1.shape[0]):
        bestMatch = -99
        for j in xrange(corners2.shape[0]):
            ncc_score = ncc_match(img1, img2, corners1[i], corners2[j],
R) # get the matching scores for selected corners
            if ncc_score > best_match[i]:
                bestMatch = j
                best_match[i] = ncc_score
        if bestMatch != -99:
            matching.append((corners1[i], corners2[bestMatch]))
    return matching
```

In [227]:
```
# detect corners on warrior and matrix sets
# adjust your corner detection parameters here
nCorners = 30
smoothSTD = 2
windowSize = 11

# read images and detect corners on images
imgs_mat = []
crns_mat = []
imgs_war = []
crns_war = []
for i in range(2):
    img_mat = imread('p4/matrix/matrix' + str(i) + '.png')
    imgs_mat.append(rgb2gray(img_mat))
    # downsize your image in case corner_detect runs slow in test
    # imgs_mat.append(rgb2gray(img_mat)[::2, ::2])
    crns_mat.append(corner_detect(imgs_mat[i], nCorners, smoothSTD, wind
owSize))

    img_war = imread('p4/warrior/warrior' + str(i) + '.png')
    imgs_war.append(rgb2gray(img_war))
    # imgs_war.append(rgb2gray(img_war)[::2, ::2])
    crns_war.append(corner_detect(imgs_war[i], nCorners, smoothSTD, wind
owSize))
```

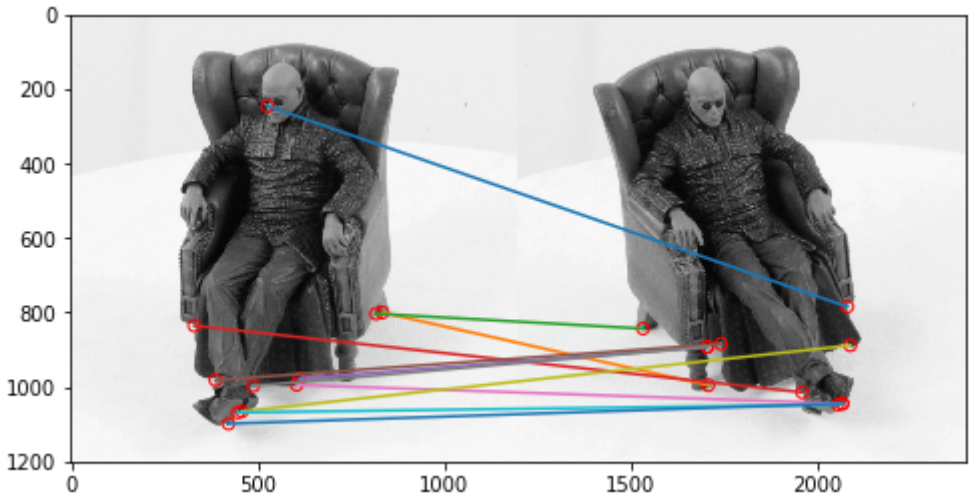
```
/Users/huangzhisheng/anaconda2/lib/python2.7/site-packages/ipykernel_la
uncher.py:13: DeprecationWarning: `imread` is deprecated!
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.
  del sys.path[0]
/Users/huangzhisheng/anaconda2/lib/python2.7/site-packages/ipykernel_la
uncher.py:19: DeprecationWarning: `imread` is deprecated!
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.
```

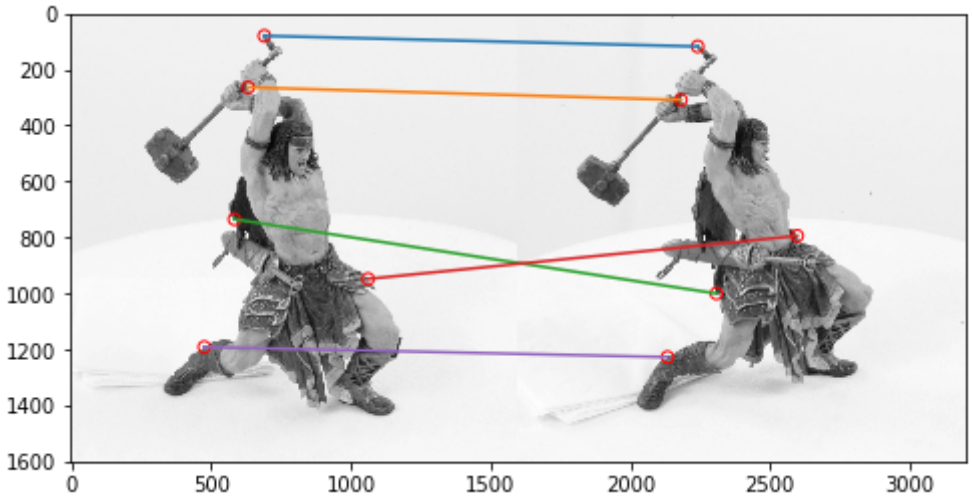
In [231]:
```
# match corners
R = 15
NCCth = 0.85
matching_mat = naive_matching(imgs_mat[0]/255, imgs_mat[1]/255, crns_mat
[0], crns_mat[1], R, NCCth)
matching_war = naive_matching(imgs_war[0]/255, imgs_war[1]/255, crns_war
[0], crns_war[1], R, NCCth)
```

In [232]:
```
print(matching_mat)
```

```
[(array([527., 246.]), array([882., 784.])), (array([835., 800.]), arra
y([509., 995.])), (array([818., 803.]), array([334., 843.])), (array([3
30., 835.]), array([ 761., 1014.])), (array([613., 977.]), array([506.,
892.])), (array([387., 981.]), array([543., 884.])), (array([605., 99
4.]), array([ 869., 1044.])), (array([488., 997.]), array([506., 89
2.])), (array([ 458., 1065.]), array([890., 888.])), (array([ 447., 106
8.]), array([ 857., 1047.])), (array([ 422., 1098.]), array([ 869., 104
4.]))]
```

In [233]:
```
# plot matching result
def show_matching_result(img1, img2, matching):
    fig = plt.figure(figsize=(8, 8))
    plt.imshow(np.hstack((img1, img2)), cmap='gray') # two dino images a
re of different sizes, resize one before use
    for p1, p2 in matching:
        plt.scatter(p1[0], p1[1], s=35, edgecolors='r', facecolors='non
e')
        plt.scatter(p2[0] + img1.shape[1], p2[1], s=35, edgecolors='r',
facecolors='none')
        plt.plot([p1[0], p2[0] + img1.shape[1]], [p1[1], p2[1]])
    plt.savefig('dino_matching.png')
    plt.show()

show_matching_result(imgs_mat[0], imgs_mat[1], matching_mat)
show_matching_result(imgs_war[0], imgs_war[1], matching_war)
```

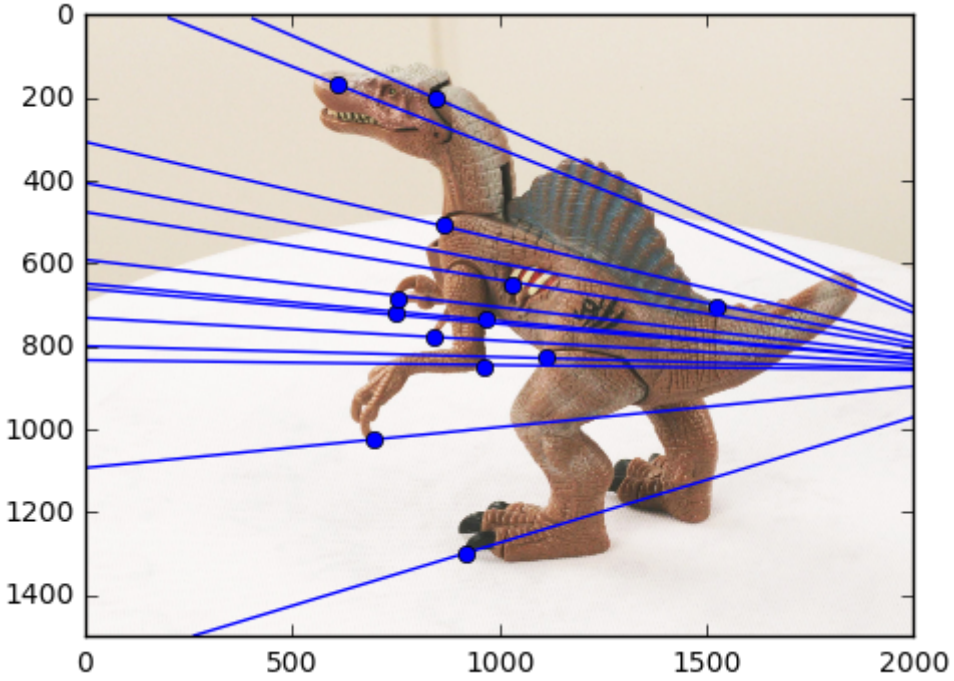


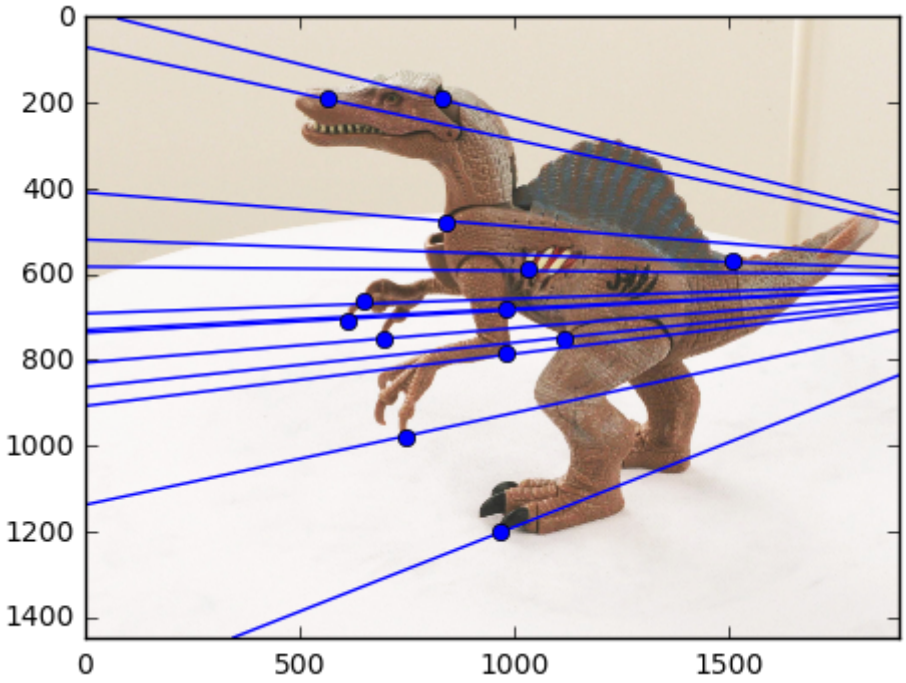


Epipolar Geometry [4 pts]

Using the fundamental_matrix function, and the corresponding points provided in cor1.npy and cor2.npy, calculate the fundamental matrix.

Using this fundamental matrix, plot the epipolar lines in both image pairs across all images. For this part you may want to complete the function plot_epipolar_lines. Shown your result for matrix and warrior as the figure below.





Also, write the script to calculate the epipoles for a given Fundamental matrix and corner point correspondences in the two images.

In [16]:
```
import numpy as np
from scipy.misc import imread
import matplotlib.pyplot as plt
from scipy.io import loadmat

def compute_fundamental(x1,x2):
    """    Computes the fundamental matrix from corresponding points
        (x1,x2 3*n arrays) using the 8 point algorithm.
        Each row in the A matrix below is constructed as
        [x'*x, x'*y, x', y'*x, y'*y, y', x, y, 1]
    """

    n = x1.shape[1]
    if x2.shape[1] != n:
        raise ValueError("Number of points don't match.")

    # build matrix for equations
    A = np.zeros((n,9))
    for i in range(n):
        A[i] = [x1[0,i]*x2[0,i], x1[0,i]*x2[1,i], x1[0,i]*x2[2,i],
                x1[1,i]*x2[0,i], x1[1,i]*x2[1,i], x1[1,i]*x2[2,i],
                x1[2,i]*x2[0,i], x1[2,i]*x2[1,i], x1[2,i]*x2[2,i] ]

    # compute linear least square solution
    U,S,V = np.linalg.svd(A)
    F = V[-1].reshape(3,3)

    # constrain F
    # make rank 2 by zeroing out last singular value
    U,S,V = np.linalg.svd(F)
    S[2] = 0
    F = np.dot(U,np.dot(np.diag(S),V))

    return F/F[2,2]


def fundamental_matrix(x1,x2):
    n = x1.shape[1]
    if x2.shape[1] != n:
        raise ValueError("Number of points don't match.")

    # normalize image coordinates
    x1 = x1 / x1[2]
    mean_1 = np.mean(x1[:2],axis=1)
    S1 = np.sqrt(2) / np.std(x1[:2])
    T1 = np.array([[S1,0,-S1*mean_1[0]],[0,S1,-S1*mean_1[1]],[0,0,1]])
    x1 = np.dot(T1,x1)

    x2 = x2 / x2[2]
    mean_2 = np.mean(x2[:2],axis=1)
    S2 = np.sqrt(2) / np.std(x2[:2])
    T2 = np.array([[S2,0,-S2*mean_2[0]],[0,S2,-S2*mean_2[1]],[0,0,1]])
    x2 = np.dot(T2,x2)

    # compute F with the normalized coordinates
    F = compute_fundamental(x1,x2)
```

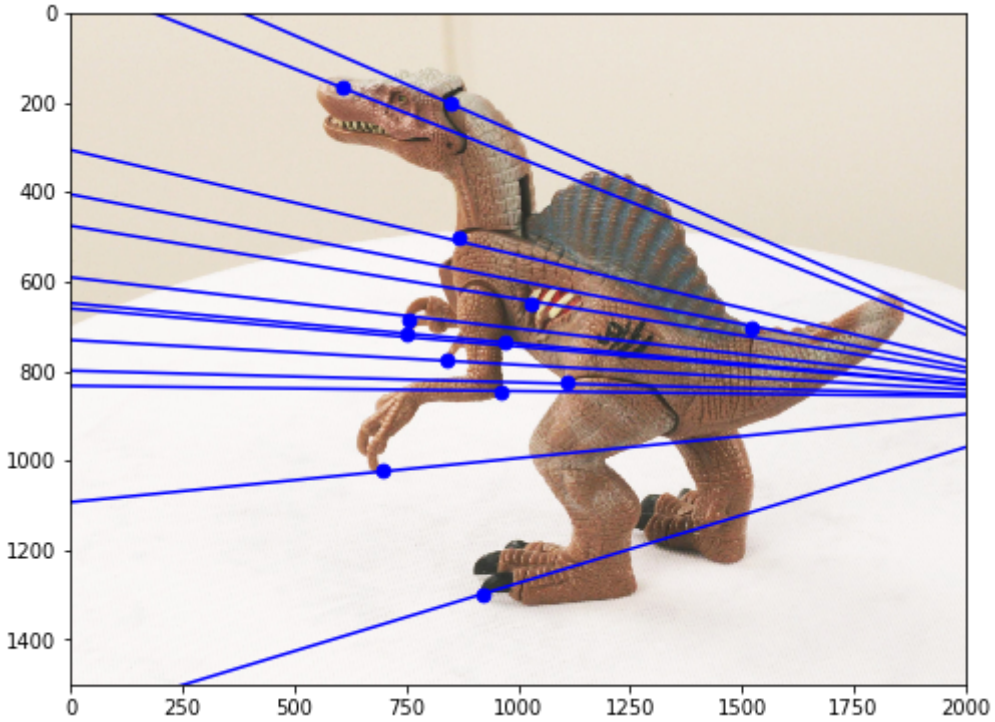
```
    # reverse normalization
    F = np.dot(T1.T,np.dot(F,T2))

    return F/F[2,2]
def compute_epipole(F):
    '''
    This function computes the epipoles for a given fundamental matrix a
nd corner point correspondences
    input:
    F--> Fundamental matrix
    output:
    e1--> corresponding epipole in image 1
    e2--> epipole in image2
    '''
    #your code here
    w, v = np.linalg.eig(F)
    res = np.vstack((w, v))
    diff = abs(res[0, :] - 0)
    a = np.where(diff == diff.min())
    e2 = res[:, a][1:]
    e2.shape = (3, 1)
    e2 = e2/e2[2]


    w, v = np.linalg.eig(F.T)
    res = np.vstack((w, v))
    diff = abs(res[0, :] - 0)
    a = np.where(diff == diff.min())
    e1 = res[:, a][1:]
    e1.shape = (3, 1)
    e1 = e1/e1[2]

    return e1, e2
```

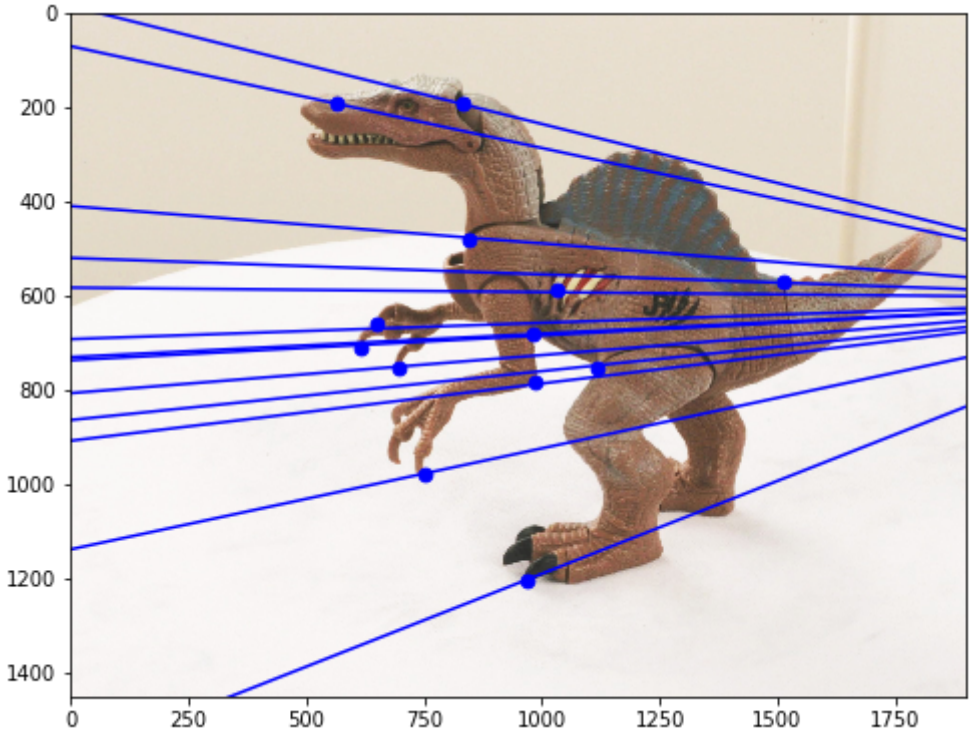
In [64]:
```
def plot_epipolar_lines(img1,img2, cor1, cor2):
    """Plot epipolar lines on image given image, corners

    Args:
        img1: Image 1.
        img2: Image 2.
        cor1: Corners in homogeneous image coordinate in image 1 (3xn)
        cor2: Corners in homogeneous image coordinate in image 2 (3xn)

    """

    """
    Your code here:
    """

    F = fundamental_matrix(cor1, cor2) # obtain the fundamental matrix
    e1, e2 = compute_epipole(F) # get the epipoles for image I and Image
 II
    #print(e1)

    ## on image I
    fig = plt.figure(figsize = (8,8))
    plt.imshow(img1, cmap = 'gray')

    for i in xrange(cor2.shape[1]): # L1 : aq_1 = 0, where a = dot(F, q_
2)
        L1 = np.dot(F, cor2[:, i]) # z = 1

        x1_end = img1.shape[1]
        x1_start = 0#e1[0]
        y1_start = - L1[2] / L1[1]#e1[1]
        y1_end = -(x1_end * L1[0] + L1[2]) / L1[1]
        #plt.axis([0, img1.shape[1], 0, img1.shape[0]])
        plt.axis([0, img1.shape[1], img1.shape[0], 0])
        plt.plot([x1_start, x1_end], [y1_start, y1_end], color = 'b')
    for i in xrange(cor1.shape[1]):
        plt.scatter(cor1[0][i], cor1[1][i], s=45, edgecolors='b', faceco
lors='b') # plot the corners on image
    plt.show()

    fig = plt.figure(figsize=(8, 8))
    plt.imshow(img2, cmap = 'gray')
    for i in range(cor1.shape[1]): # L2:  b q_2 = 0, where b = dot(F.T,
 q_1)
        L2 = np.dot(F.T, cor1[:,i]) # z = 1
        x2_start = 0
        x2_end = img2.shape[1]
        y2_start = -L2[2]/(L2[1])
        y2_end = -(x2_end * L2[0] + L2[2]) / (L2[1])
        plt.axis([0,img2.shape[1], img2.shape[0],0])
        plt.plot([x2_start, x2_end], [y2_start, y2_end], color = 'b')
    for i in range(cor2.shape[1]):
        plt.scatter(cor2[0][i], cor2[1][i], s=45, edgecolors='b', faceco
lors='b')
```

```
    plt.show()
```

In [190]:
```
# replace images and corners with those of matrix and warrior
I1 = imread("./p4/dino/dino0.png")
I2 = imread("./p4/dino/dino1.png")

cor1 = np.load("./p4/dino/cor1.npy")
cor2 = np.load("./p4/dino/cor2.npy")

I3 = imread("./p4/warrior/warrior0.png")
I4 = imread("./p4/warrior/warrior1.png")

cor3 = np.load("./p4/warrior/cor1.npy")
cor4 = np.load("./p4/warrior/cor2.npy")

I5 = imread("./p4/matrix/matrix0.png")
I6 = imread("./p4/matrix/matrix1.png")

cor5 = np.load("./p4/matrix/cor3.npy")
cor6 = np.load("./p4/matrix/cor4.npy")



plot_epipolar_lines(I1,I2,cor1,cor2)
#print(cor6[:,1])
```

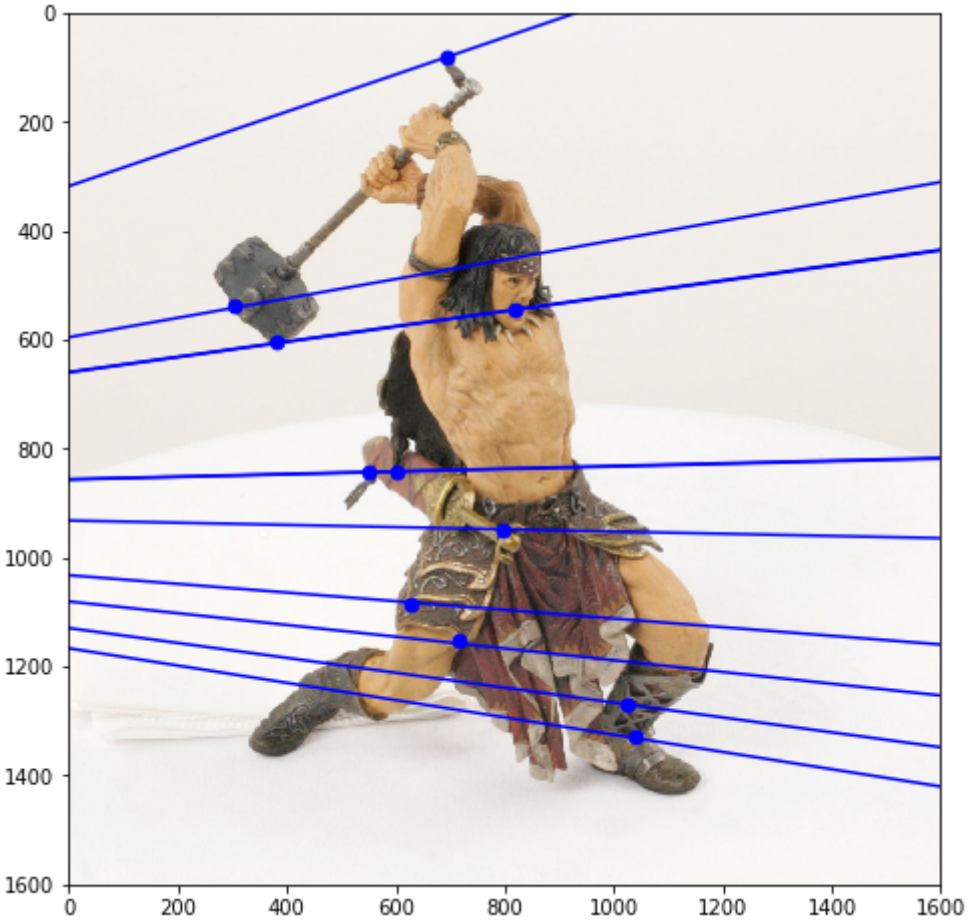
```
/Users/huangzhisheng/anaconda2/lib/python2.7/site-packages/ipykernel_la
uncher.py:2: DeprecationWarning: `imread` is deprecated!
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.

/Users/huangzhisheng/anaconda2/lib/python2.7/site-packages/ipykernel_la
uncher.py:3: DeprecationWarning: `imread` is deprecated!
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.
  This is separate from the ipykernel package so we can avoid doing imp
orts until
/Users/huangzhisheng/anaconda2/lib/python2.7/site-packages/ipykernel_la
uncher.py:8: DeprecationWarning: `imread` is deprecated!
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.

/Users/huangzhisheng/anaconda2/lib/python2.7/site-packages/ipykernel_la
uncher.py:9: DeprecationWarning: `imread` is deprecated!
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.
  if __name__ == '__main__':
/Users/huangzhisheng/anaconda2/lib/python2.7/site-packages/ipykernel_la
uncher.py:14: DeprecationWarning: `imread` is deprecated!
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.

/Users/huangzhisheng/anaconda2/lib/python2.7/site-packages/ipykernel_la
uncher.py:15: DeprecationWarning: `imread` is deprecated!
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.
  from ipykernel import kernelapp as app
```

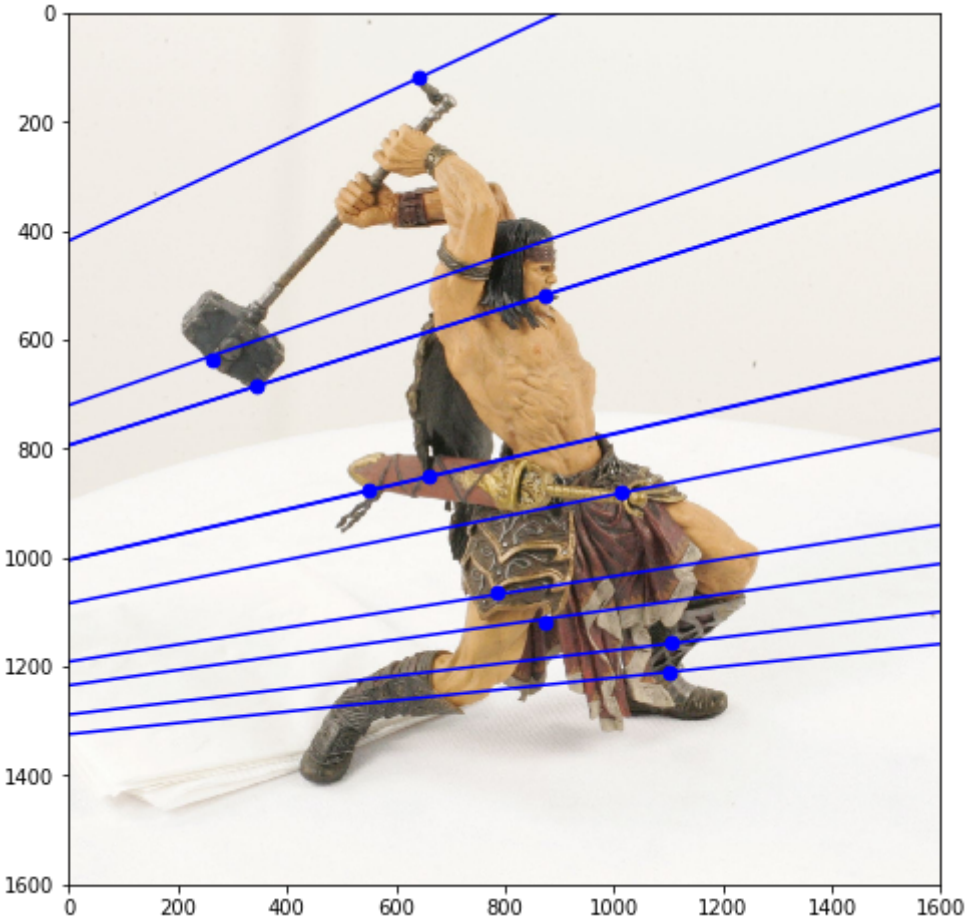



0
200
400
600
800
1000
1200
1400
0
250
500
750
1000
1250
1500
1750

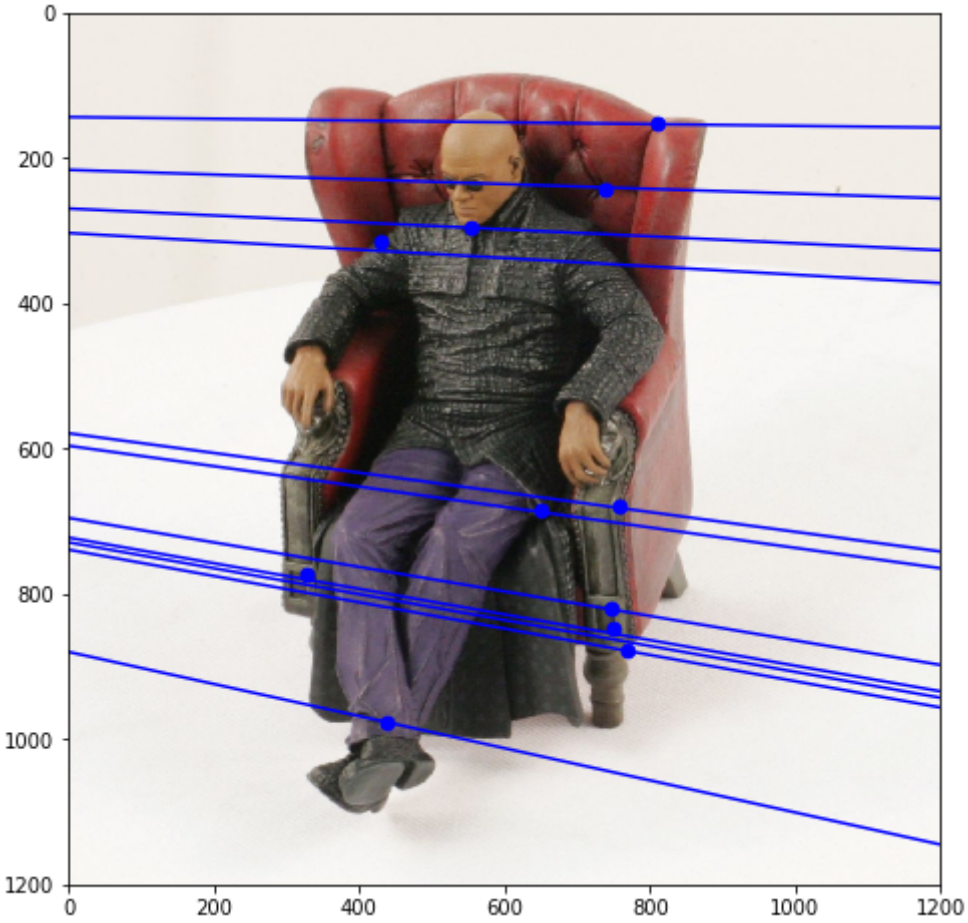
```
[536. 295.   1.]
```

In [191]:
```
plot_epipolar_lines(I3,I4,cor3,cor4)
```


0
200
400
600
800
1000
1200
1400
1600
0
200
400
600
800
1000
1200
1400
1600


0
200
400
600
800
1000
1200
1400
1600
0
200
400
600
800
1000
1200
1400
1600

In [192]:
```
plot_epipolar_lines(I5,I6,cor5,cor6)
```


0
200
400
600
800
1000
1200
0
200
400
600
800
1000
1200

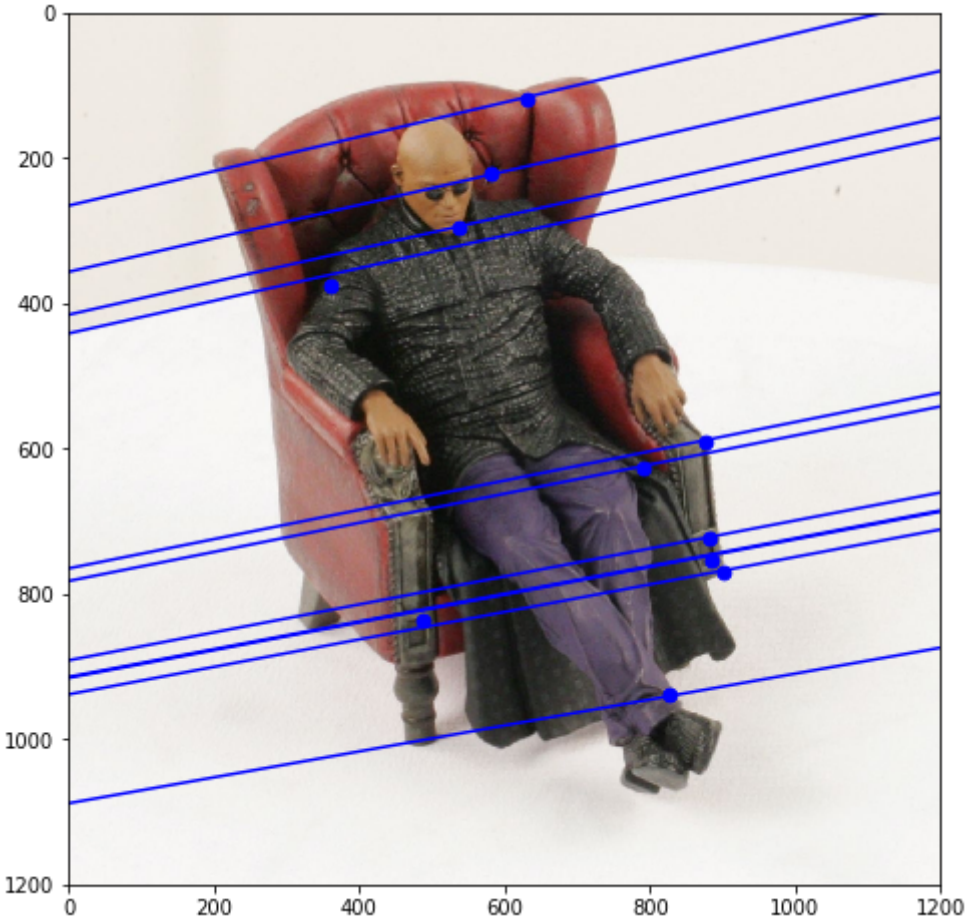
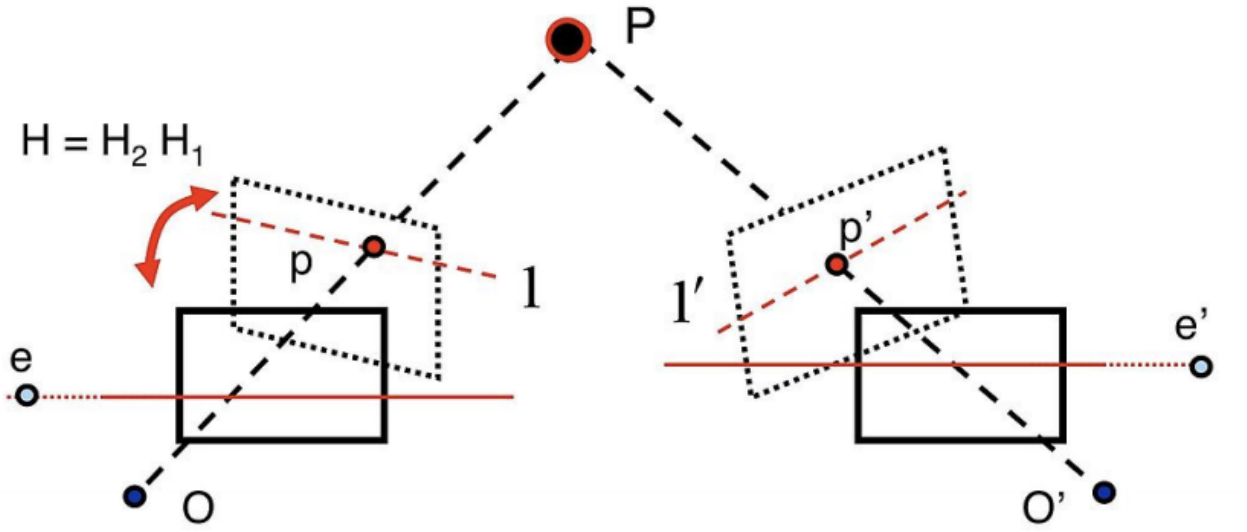
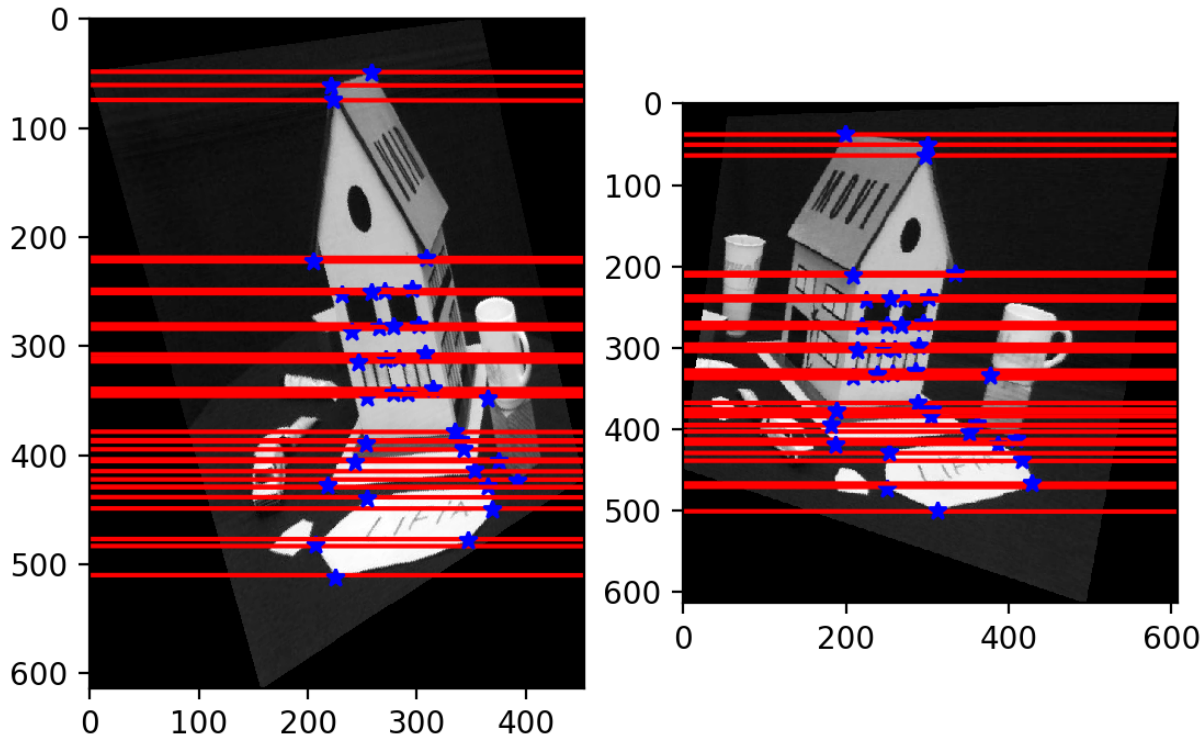

0
200
400
600
800
1000
1200
0
200
400
600
800
1000
1200

Image Rectification [3 pts]

An interesting case for epipolar geometry occurs when two images are parallel to each other. In this case, there is no rotation component involved between the two images and the essential matrix is $\mathbb{E} = [\boldsymbol{T_x}]\boldsymbol{R} = [\boldsymbol{T_x}]$. Also if you observe the epipolar lines $\boldsymbol{l}$ and $\boldsymbol{l'}$ for parallel images, they are horizontal and consequently, the corresponding epipolar lines share the same vertical coordinate. Therefore the process of making images parallel becomes useful while discerning the relationships between corresponding points in images. Rectifying a pair of images can also be done for uncalibrated camera images (i.e. we do not require the camera matrix of intrinsic parameters). Using the fundamental matrix we can find the pair of epipolar lines $\boldsymbol{l_i}$ and $\boldsymbol{l_i'}$ for each of the correspondances. The intersection of these lines will give us the respective epipoles $\boldsymbol{e}$ and $\boldsymbol{e'}$. Now to make the epipolar lines to be parallel we need to map the epipoles to infinity. Hence , we need to find a homography that maps the epipoles to infinity. The method to find the homography has been implemented for you. You can read more about the method used to estimate the homography in the paper "Theory and Practice of Projective Rectification" by Richard Hartley.



Using the compute_epipoles function from the previous part and the given compute_matching_homographies function, find the rectified images and plot the parallel epipolar lines using the plot_epipolar_lines function from above. You need to do this for both the matrix and the warrior images. A sample output will look as below:


0
100
200
300
400
500
600
0
100
200
300
400
0
100
200
300
400
500
600
0
200
400
600

In [241]:
```
def compute_matching_homographies(e2, F, im2, points1, points2):

    '''This function computes the homographies to get the rectified imag
es
    input:
    e2--> epipole in image 2
    F--> the Fundamental matrix
    im2--> image2
    points1 --> corner points in image1
    points2--> corresponding corner points in image2
    output:
    H1--> Homography for image 1
    H2--> Homography for image 2
    '''
    # calculate H2
    width = im2.shape[1]
    height = im2.shape[0]

    T = np.identity(3)
    T[0][2] = -1.0 * width / 2
    T[1][2] = -1.0 * height / 2

    e = T.dot(e2)
    e1_prime = e[0]
    e2_prime = e[1]
    if e1_prime >= 0:
        alpha = 1.0
    else:
        alpha = -1.0

    R = np.identity(3)
    R[0][0] = alpha * e1_prime / np.sqrt(e1_prime**2 + e2_prime**2)
    R[0][1] = alpha * e2_prime / np.sqrt(e1_prime**2 + e2_prime**2)
    R[1][0] = - alpha * e2_prime / np.sqrt(e1_prime**2 + e2_prime**2)
    R[1][1] = alpha * e1_prime / np.sqrt(e1_prime**2 + e2_prime**2)

    f = R.dot(e)[0]
    G = np.identity(3)
    G[2][0] = - 1.0 / f

    H2 = np.linalg.inv(T).dot(G.dot(R.dot(T)))

    # calculate H1
    e_prime = np.zeros((3, 3))
    e_prime[0][1] = -e2[2]
    e_prime[0][2] = e2[1]
    e_prime[1][0] = e2[2]
    e_prime[1][2] = -e2[0]
    e_prime[2][0] = -e2[1]
    e_prime[2][1] = e2[0]

    v = np.array([1, 1, 1])
    M = e_prime.dot(F) + np.outer(e2, v)

    points1_hat = H2.dot(M.dot(points1.T)).T
    points2_hat = H2.dot(points2.T).T
```

```
    W = points1_hat / points1_hat[:, 2].reshape(-1, 1)
    b = (points2_hat / points2_hat[:, 2].reshape(-1, 1))[:, 0]

    # least square problem
    a1, a2, a3 = np.linalg.lstsq(W, b)[0]
    HA = np.identity(3)
    HA[0] = np.array([a1, a2, a3])

    H1 = HA.dot(H2).dot(M)
    return H1, H2

def image_rectification(im1,im2,points1,points2):
    '''this function provides the rectified images along with the new co
rner points as outputs for a given pair of
    images with corner correspondences
    input:
    im1--> image1
    im2--> image2
    points1--> corner points in image1
    points2--> corner points in image2
    output:
    rectified_im1-->rectified image 1
    rectified_im2-->rectified image 2
    new_cor1--> new corners in the rectified image 1
    new_cor2--> new corners in the rectified image 2
    '''
    "your code here"
    F = fundamental_matrix(points1, points2) # obtain the fundamental ma
trix
    e1, e2 = compute_epipole(F) # get the epipoles for image I and Image
 II
    H1, H2 = compute_matching_homographies(e2, F.T, im2, points1.T, poin
ts2.T) # get the homegraphis

    pt1 = np.array([[0,0],[im1.shape[1],0],[im1.shape[1],im1.shape[0]],[
0,im1.shape[0]]]).T
    pt2 = np.array([[0,0],[im2.shape[1],0],[im2.shape[1],im2.shape[0]],[
0,im2.shape[0]]]).T
    target_points1 = from_homog(np.matmul(H1, to_homog(pt1)))
    target_points2 = from_homog(np.matmul(H2, to_homog(pt2)))

    im1_size1 = int(target_points1[0,:].max() - target_points1[0, :].min
())
    im1_size2 = int(target_points1[1,:].max() - target_points1[1, :].min
())
    im1_size = np.max((im1_size1, im1_size2))
    im2_size1 = int(target_points2[0,:].max() - target_points2[0, :].min
())
    im2_size2 = int(target_points2[1,:].max() - target_points2[1, :].min
())
    # target_image1 = np.zeros((im1_size1, im1_size2, 3))
    target_image1 = np.ones((im1.shape[1], im1.shape[0], 3))
    target_image2 = np.ones((im2.shape[1], im2.shape[0], 3))

    rectified_im1 = warp3(target_image1, target_points1, im1, np.linalg.
inv(H1))
```

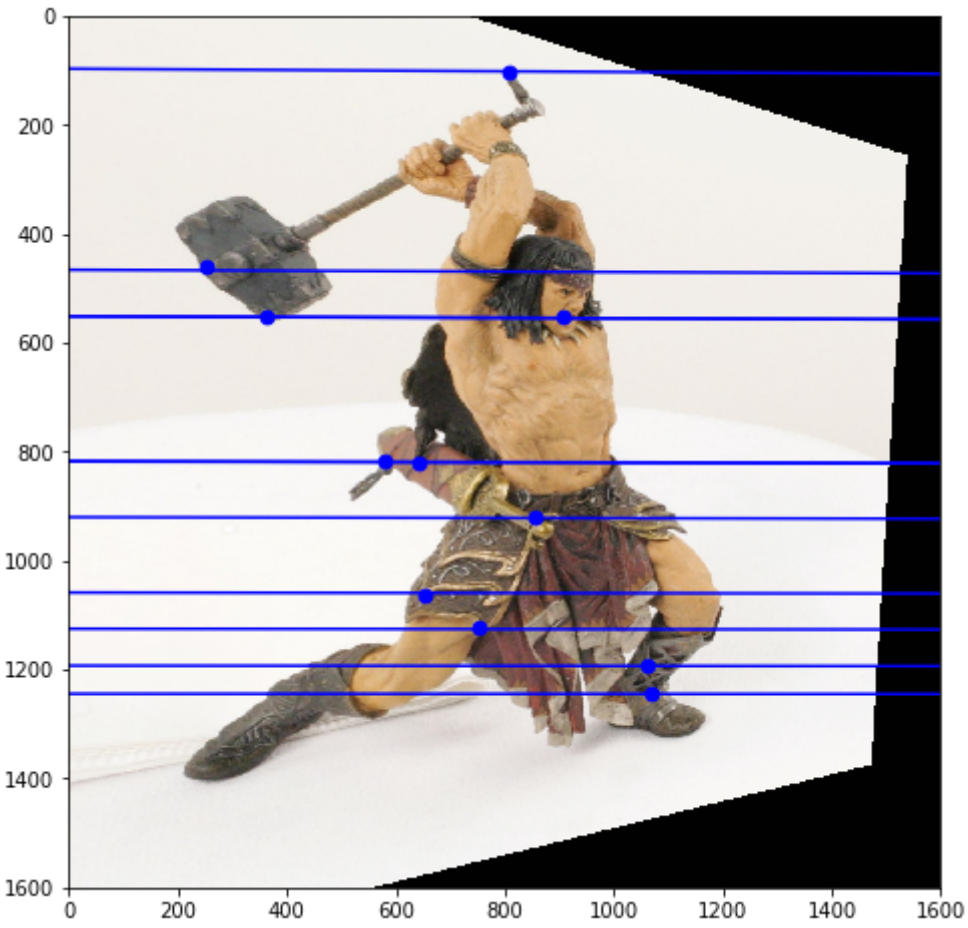
```
    rectified_im2 = warp3(target_image2, target_points2, im2, np.linalg.
inv(H2))
    new_cor1 = np.matmul(H1, points1)
    new_cor2 = np.matmul(H2, points2)

    return rectified_im1,rectified_im2,new_cor1/new_cor1[2],new_cor2/new
_cor2[2]


# convert points from euclidian to homogeneous
def to_homog(points):
    if len(points) == 0 or points is None:
        return None

    if len(points.shape) == 1:
        oneLine = np.array([1]).T # add '1'
        return np.append(points, oneLine)
    else:
        oneLine = np.ones(points.shape[1])
        return np.vstack((points, oneLine)) # add a line of '1's if usin
g matrices

###dfadfa
def from_homog(points_homog):
    if len(points_homog) == 0 or points_homog is None:
        return None

    return points_homog[:-1]/points_homog[-1] # [x/z, y/z]

#source_image2 = imread('ucsd_logo.png')[:,:,:3]/255.

def warp3(target_image, target_points, source_image, H):
 # Selecting the four points on the source image
    source_size = source_image.shape
    source_points = np.array([[0,0],[source_size[1],0],[source_size[1],s
ource_size[0]],[0,source_size[0]]]).T
    # Target points from previous cells, we choose to inverse mapping be
cause
    #of our demonstration
    # from warp1 and warp2 that inverse mapping is the better approach
    Hinv = H#computeH(to_homog(target_points), to_homog(source_points))
    # Fill in the values in the target image form source image
    for i in range(target_image.shape[1]):
        for j in range(target_image.shape[0]):
        # Transformed points after applying homography
            new_points = from_homog(np.matmul(Hinv,to_homog(np.array([[i
,j]]).T)))
            y,x = int(new_points[0]),int(new_points[1])
            # Check if point exists in source image
            if x>=0 and x<source_size[1] and y>=0 and y<source_size[0]:
                target_image[j][i] = source_image[x][y]

                #print(source_image[x][y])
    return target_image
```

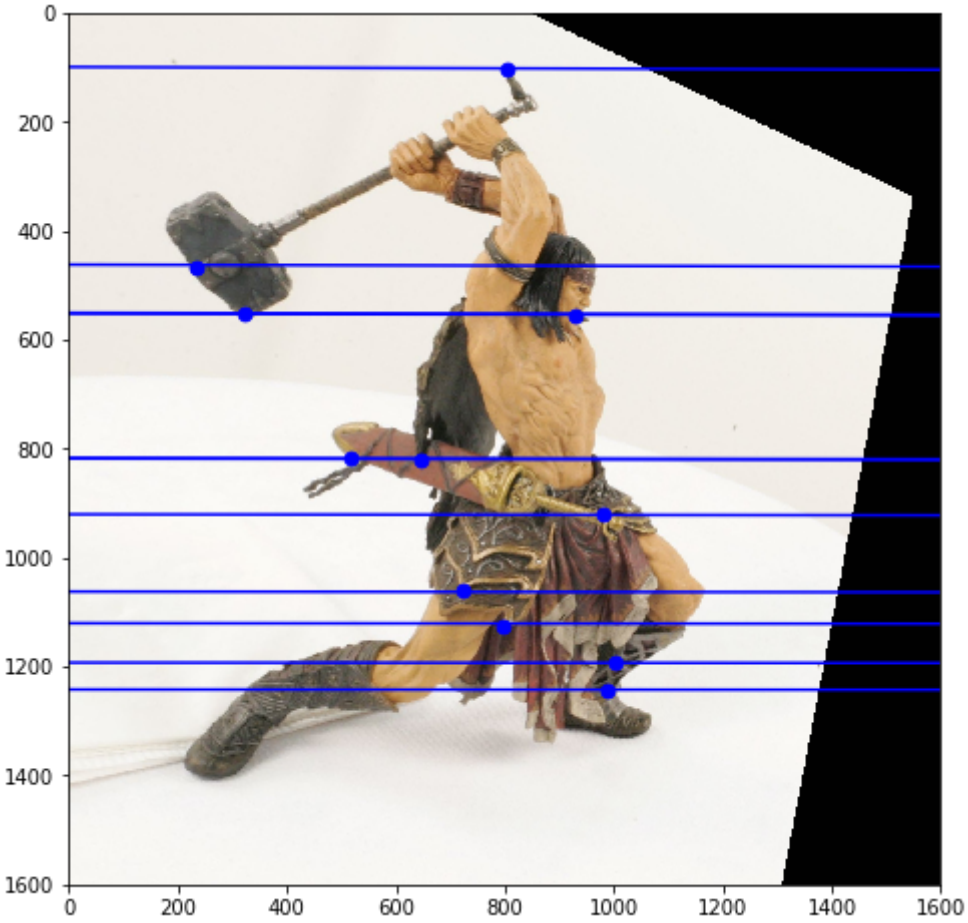
In [242]:
```
I3 = imread("./p4/warrior/warrior0.png")
I4 = imread("./p4/warrior/warrior1.png")

cor3 = np.load("./p4/warrior/cor1.npy")
cor4 = np.load("./p4/warrior/cor2.npy")


I5 = imread("./p4/matrix/matrix0.png")
I6 = imread("./p4/matrix/matrix1.png")

cor5 = np.load("./p4/matrix/cor3.npy")
cor6 = np.load("./p4/matrix/cor4.npy")
```

```
/Users/huangzhisheng/anaconda2/lib/python2.7/site-packages/ipykernel_la
uncher.py:1: DeprecationWarning: `imread` is deprecated!
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.
  """Entry point for launching an IPython kernel.
/Users/huangzhisheng/anaconda2/lib/python2.7/site-packages/ipykernel_la
uncher.py:2: DeprecationWarning: `imread` is deprecated!
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.

/Users/huangzhisheng/anaconda2/lib/python2.7/site-packages/ipykernel_la
uncher.py:8: DeprecationWarning: `imread` is deprecated!
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.

/Users/huangzhisheng/anaconda2/lib/python2.7/site-packages/ipykernel_la
uncher.py:9: DeprecationWarning: `imread` is deprecated!
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.
  if __name__ == '__main__':
```

In [243]:
```
war_rectified_im1, war_rectified_im2, war_cor3, war_cor4 = image_rectifi
cation(I3,I4,cor3,cor4)
plot_epipolar_lines(war_rectified_im1/255.0, war_rectified_im2/255.0, wa
r_cor3, war_cor4)
```

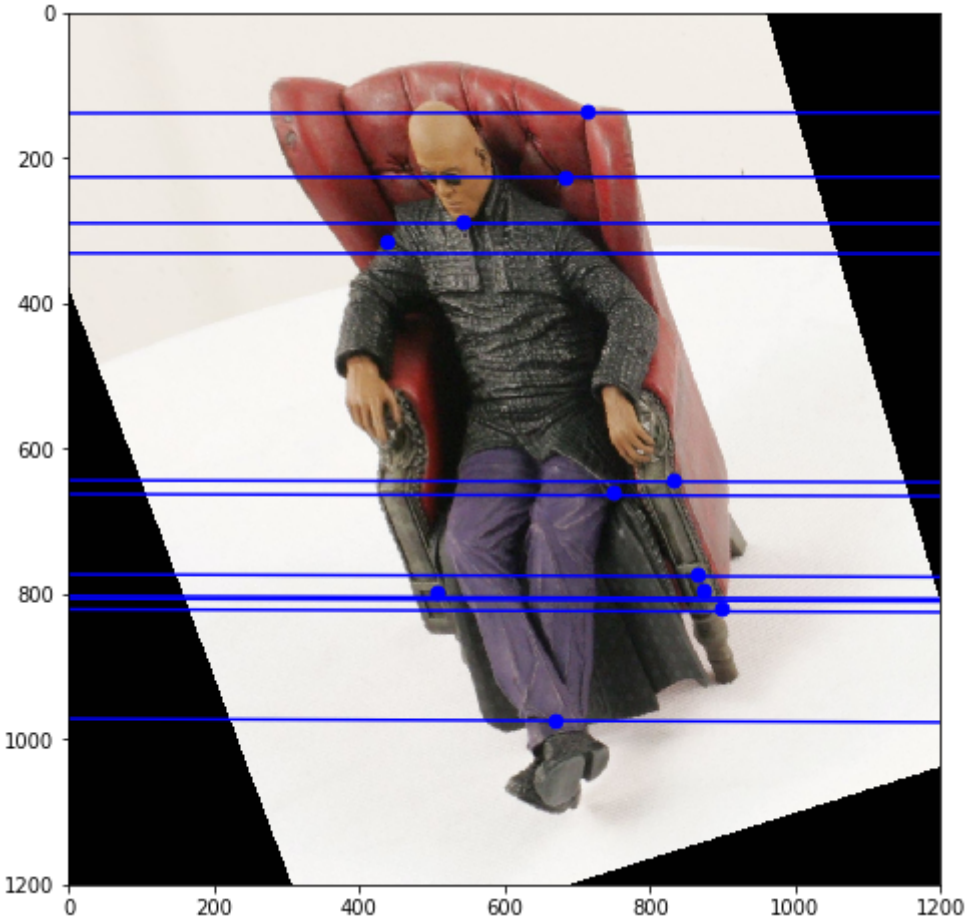
```
/Users/huangzhisheng/anaconda2/lib/python2.7/site-packages/ipykernel_la
uncher.py:61: FutureWarning: `rcond` parameter will change to the defau
lt of machine precision times ``max(M, N)`` where M and N are the input
matrix dimensions.
To use the future default and silence this warning we advise to pass `r
cond=None`, to keep using the old, explicitly pass `rcond=-1`.
```

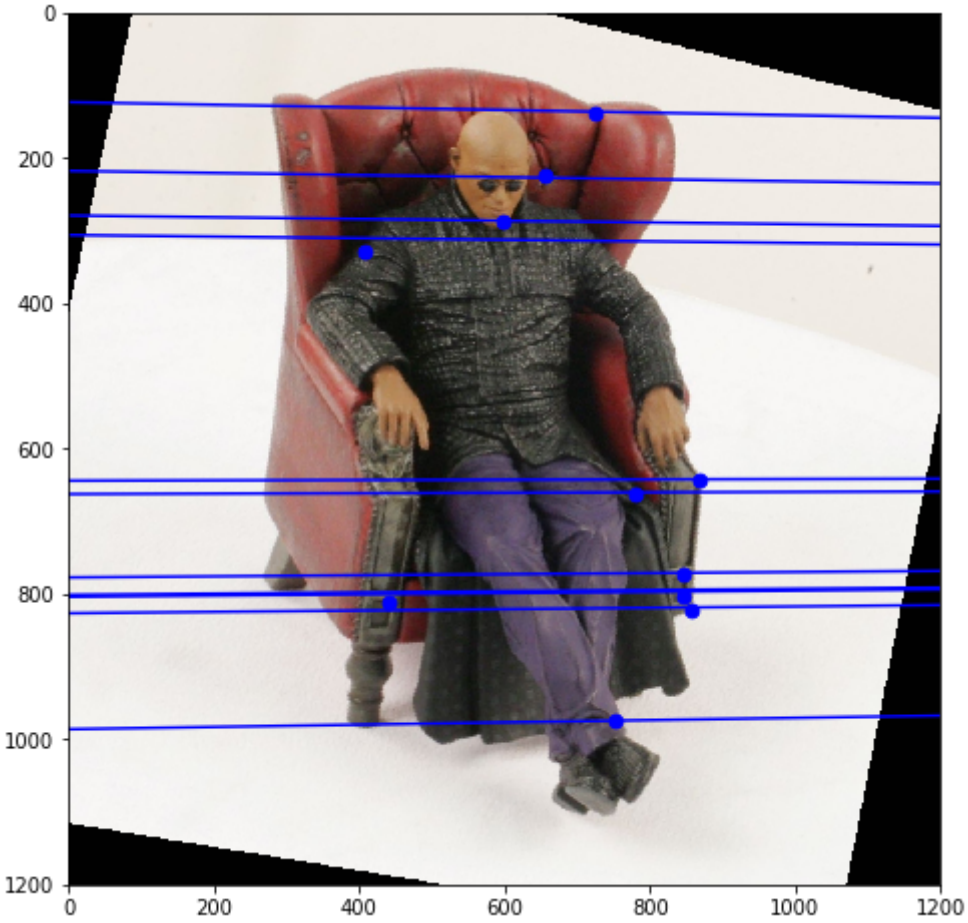



0
200
400
600
800
1000
1200
1400
1600
0
200
400
600
800
1000
1200
1400
1600

In [244]:
```
mat_rectified_im1, mat_rectified_im2, mat_cor5, mat_cor6 = image_rectifi
cation(I5,I6,cor5,cor6)
plot_epipolar_lines(mat_rectified_im1/255.0, mat_rectified_im2/255.0, ma
t_cor5, mat_cor6)
```

```
/Users/huangzhisheng/anaconda2/lib/python2.7/site-packages/ipykernel_la
uncher.py:61: FutureWarning: `rcond` parameter will change to the defau
lt of machine precision times ``max(M, N)`` where M and N are the input
matrix dimensions.
To use the future default and silence this warning we advise to pass `r
cond=None`, to keep using the old, explicitly pass `rcond=-1`.
```





Matching Using epipolar geometry[4 pts]

We will now use the epipolar geometry constraint on the rectified images and updated corner points to build a better matching algorithm. First, detect 10 corners in Image1. Then, for each corner, do a linesearch along the corresponding parallel epipolar line in Image2. Evaluate the NCC score for each point along this line and return the best match (or no match if all scores are below the NCCth). R is the radius (size) of the NCC patch in the code below. You do not have to run this in both directions. Show your result as in the naive matching part. Execute this for the warrior and matrix images.

In [348]:
```
import sys
def display_correspondence(img1, img2, corrs):
    """Plot matching result on image pair given images and correspondenc
es

    Args:
        img1: Image 1.
        img2: Image 2.
        corrs: Corner correspondence

    """

    """
    Your code here.
    You may refer to the show_matching_result function
    """
    show_matching_result(img1, img2, corrs)

def correspondence_matching_epipole(img1, img2, corners1, F, R, NCCth):
    """Find corner correspondence along epipolar line.

    Args:
        img1: Image 1.
        img2: Image 2.
        corners1: Detected corners in image 1.
        F: Fundamental matrix calculated using given ground truth corner
 correspondences.
        R: NCC matching window radius.
        NCCth: NCC matching threshold.


    Returns:
        Matching result to be used in display_correspondence function

    """
    """
    Your code here.
    """
    cor1 = to_homog(corners1.T) # convert 2D eucludean to 3d homogenous
 cor1 nx3
    matching = []
    for i in xrange(corners1.shape[0]):
        minimum = sys.maxint
        L2 = np.dot(F.T, cor1[:,i]) # z = 1, L2 on image II
        a, b, c = L2[0], L2[1], L2[2]
        x2_end = img2.shape[1] # maximum value for x
        y2_end = img2.shape[0] # maximum value for y
        if i in xrange(R, x2_end -R):
            for j in xrange(R, x2_end - R): # x range
                y = int((-j * a - c) / b) # for each y
                if y >= R and y < y2_end - R: # test if y within the ran
ge
                    cor2_test = np.array([j, y, 1])
                    ncc_score = ncc_match(img1, img2, corners1[i,:], cor
2_test, R)
#                     print(corners[i,:])
```

```
#                     print("cor2_test", cor2_test)
                    if ncc_score < minimum:
                        minimum = ncc_score
                        cor2final = cor2_test
        if minimum <= NCCth:
            matching.append((corners1[i], cor2final))
    return matching
```

In [353]:
```
I1=imread("./p4/matrix/matrix0.png")
I2=imread("./p4/matrix/matrix1.png")
cor1 = np.load("./p4/matrix/cor1.npy")
cor2 = np.load("./p4/matrix/cor2.npy")
I3=imread("./p4/warrior/warrior0.png")
I4=imread("./p4/warrior/warrior1.png")
cor3 = np.load("./p4/warrior/cor1.npy")
cor4 = np.load("./p4/warrior/cor2.npy")
```

```
/Users/huangzhisheng/anaconda2/lib/python2.7/site-packages/ipykernel_la
uncher.py:1: DeprecationWarning: `imread` is deprecated!
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.
  """Entry point for launching an IPython kernel.
/Users/huangzhisheng/anaconda2/lib/python2.7/site-packages/ipykernel_la
uncher.py:2: DeprecationWarning: `imread` is deprecated!
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.

/Users/huangzhisheng/anaconda2/lib/python2.7/site-packages/ipykernel_la
uncher.py:5: DeprecationWarning: `imread` is deprecated!
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.
  """
/Users/huangzhisheng/anaconda2/lib/python2.7/site-packages/ipykernel_la
uncher.py:6: DeprecationWarning: `imread` is deprecated!
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.
```

In [362]:
```
rectified_im1,rectified_im2,new_cor1,new_cor2 = image_rectification(I1,I
2,cor1,cor2)
rectified_im3,rectified_im4,new_cor3,new_cor4 = image_rectification(I3,I
4,cor3,cor4)

F_new = fundamental_matrix(new_cor1, new_cor2)

nCorners = 10
#decide the NCC matching window radius
R = 5
smoothSTD = 2
windowSize = 11
NCCth = 0.50
# detect corners using corner detector here, store in corners1
corners1 = corner_detect(rgb2gray(rectified_im1), nCorners, smoothSTD, w
indowSize)
```

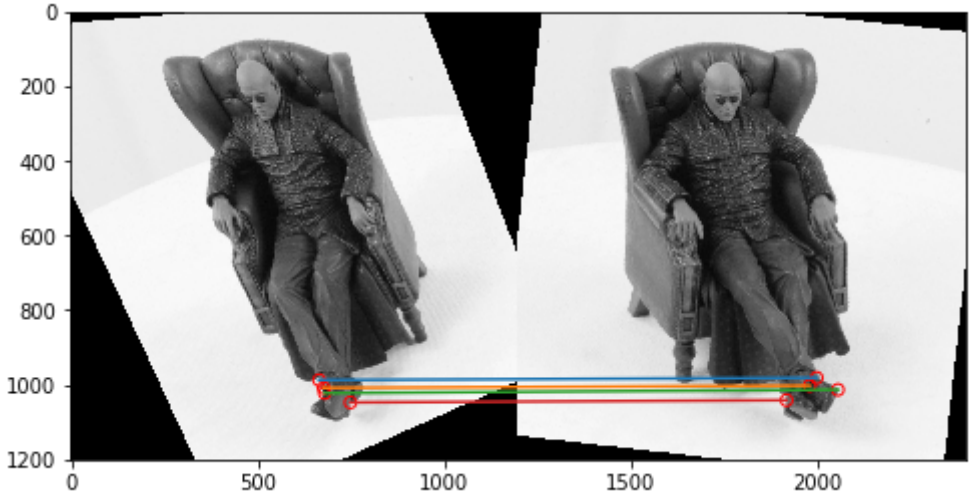
```
/Users/huangzhisheng/anaconda2/lib/python2.7/site-packages/ipykernel_la
uncher.py:61: FutureWarning: `rcond` parameter will change to the defau
lt of machine precision times ``max(M, N)`` where M and N are the input
matrix dimensions.
To use the future default and silence this warning we advise to pass `r
cond=None`, to keep using the old, explicitly pass `rcond=-1`.
```

In [363]:
```
print(corners1.shape)
corrs = correspondence_matching_epipole(rgb2gray(rectified_im1), rgb2gra
y(rectified_im2), corners1, F_new, R, NCCth)
```

```
(9, 2)
```

```
/Users/huangzhisheng/anaconda2/lib/python2.7/site-packages/ipykernel_la
uncher.py:28: RuntimeWarning: invalid value encountered in divide
```

In [364]:
```
display_correspondence(rgb2gray(rectified_im1), rgb2gray(rectified_im2),
 corrs)

print(rectified_im1.shape)
```




```
(1200, 1200, 3)
```

In [365]:
```
F_new2=fundamental_matrix(new_cor3, new_cor4)

corners2 = corner_detect(rgb2gray(rectified_im3), nCorners, smoothSTD, w
indowSize)
corrs = correspondence_matching_epipole(rgb2gray(rectified_im3), rgb2gra
y(rectified_im4), corners2, F_new2, R, NCCth)
display_correspondence(rectified_im3/255.0, rectified_im4/255.0, corrs)
```

```
/Users/huangzhisheng/anaconda2/lib/python2.7/site-packages/ipykernel_la
uncher.py:28: RuntimeWarning: invalid value encountered in divide
```