Assignment3

November 21, 2018

1 CSE 252A Computer Vision I Fall 2018 - Assignment 3

1.0.1 Instructor: David Kriegman

1.0.2 Assignment Published On: Wednesday, November 7, 2018

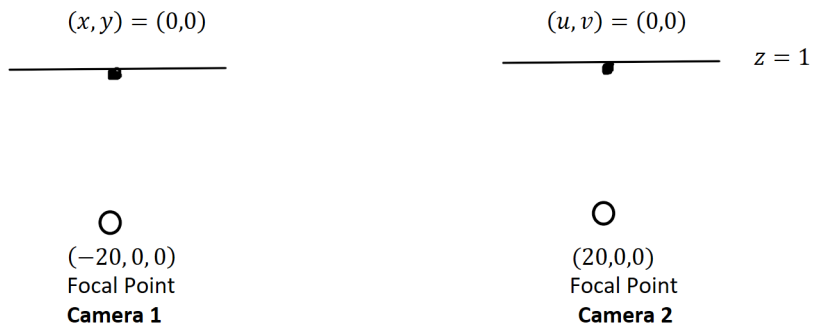
1.0.3 Due On: Tuesday, November 20, 2018 11:59 pm

1.1 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.

1.2 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.

1.3 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

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

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.

1.4 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.

1.4.1 (a)

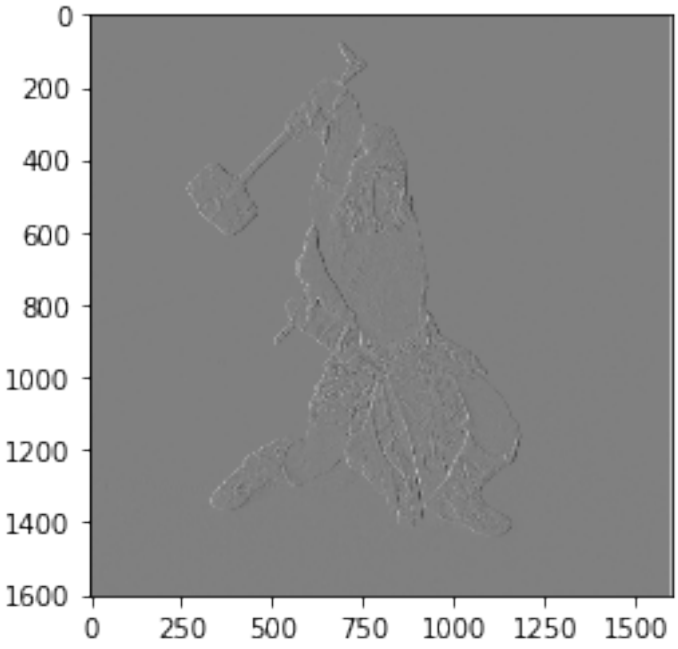
```
In [1]: import numpy as np
        from scipy.misc import imread, imresize
        import matplotlib.pyplot as plt
        from scipy.ndimage.filters import gaussian_filter

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

```
In [2]: from scipy import signal
        img_war = imread('p4/warrior/warrior' + '0' + '.png')
        img_war = rgb2gray(img_war)

        # box filter
        box = np.array(
            [[1/9, 1/9, 1/9], [1/9, 1/9, 1/9], [1/9, 1/9, 1/9]]).astype(np.float32)
        # X derivative filter
        dx = np.array([[-1, 0, 1]]).astype(np.float32)

        # first apply both one after the another
        print("Box filter and then derivate in x direction")
        img_war_b = signal.convolve2d(img_war, box, mode='full')
        img_war_bdx = signal.convolve2d(img_war_b, dx, mode='full')
        plt.imshow(img_war_bdx, cmap='gray')
        plt.show()

        # create a combined filter
        print("Combined: Box filter and derivate in x direction")
        comb_filter = signal.convolve2d(box, dx, mode='full')
        img_war_comb = signal.convolve2d(img_war, comb_filter, mode='full')
        plt.imshow(img_war_comb, cmap='gray')
        plt.show()

        print("Box filter: {}".format(box))
        print("X derivate filter: {}".format(dx))
        print("Combined filter: {}".format(comb_filter))
```

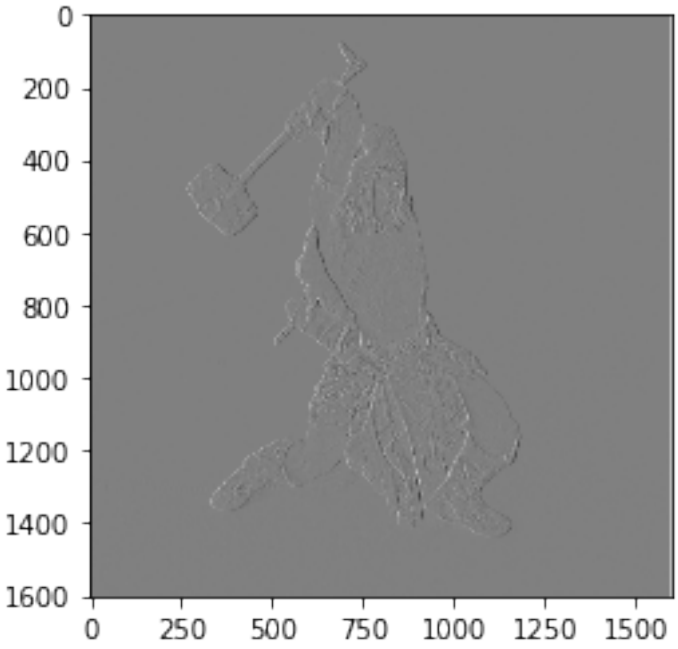
```
/home/amanraj/anaconda3/envs/neural/lib/python3.6/site-packages/ipykernel_launcher.py:2: Depre
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
```

Use ``imageio.imread`` instead.

Box filter and then derivate in x direction



Combined: Box filter and derivate in x direction




```
Box filter: [[0.11111111 0.11111111 0.11111111]
 [0.11111111 0.11111111 0.11111111]
 [0.11111111 0.11111111 0.11111111]]
X derivate filter: [[-1.  0.  1.]]
Combined filter: [[-0.11111111 -0.11111111  0.          0.11111111  0.11111111]
 [-0.11111111 -0.11111111  0.          0.11111111  0.11111111]
 [-0.11111111 -0.11111111  0.          0.11111111  0.11111111]]
```

1.4.2 (b)

A box filter used in previous part is a separable filter. It can be separated into two filters which can be applied in row and column direction. Let's take a general case where size of box filter is given by: $ (m * m) $ and convolution of such filter on image of size: $ (n * n) $ will yield total - additions: $(m^2 - 1)$ and multiplications: $(m^2 + 1)$ or in total $(2 * m^2)$ mathematical operations for each pixel. Consdiering the total number of pixels in the image, it is equal to $(2 * n^2 * m^2)$

By separting this box filter we will have two filters to convolve of size $1 * m$ for rows and $m * 1$ for columns. Convolution of each such filter on image will yield total - additions: $(m - 1)$ and multiplications: $(m + 1)$ or in total $(2 * m)$ mathematical operations for each pixel. Consdiering the total number of pixels in the image and both kernels, it is equal to $2 * (2 * n^2 * m) = 4 * n^2 * m$, which is significantly less than $(2 * n^2 * m^2)$.

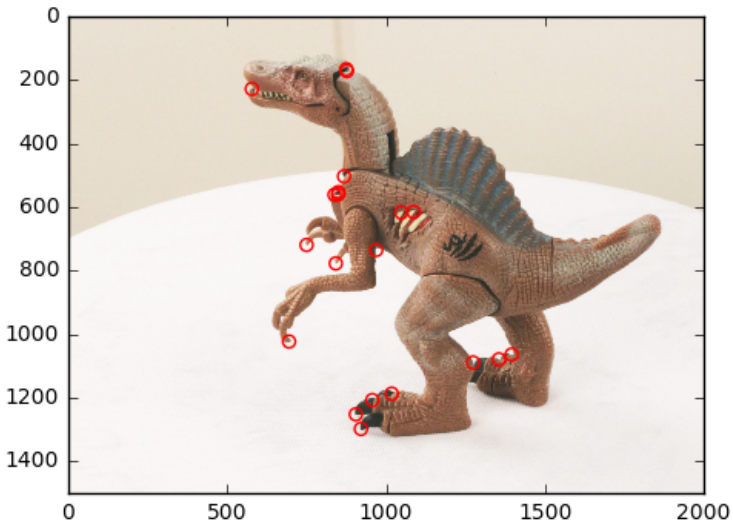
1.5 Problem 4: Sparse Stereo Matching [22 pts]

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

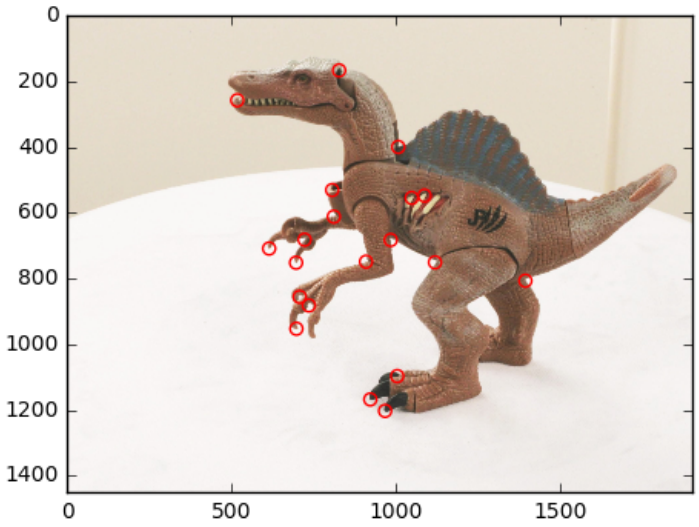
images, two camera matrices, and set sets of corresponding points (extracted by manually clicking the images). For illustration, I have run my code on a third image pair (dino1.png, dino2.png). This data is also provided for you to debug your code, but you should only report results on warrior and matrix. 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.

1.5.1 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. 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 [3]:
```
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 suppression.

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

    """
    # filter image with Gaussian first
    image = gaussian_filter(image, sigma=smoothSTD)
    # compute gradient in x and y direction
    image_dx , image_dy = np.gradient(image)

    # store min(lambda1, lambda2) for each C(x,y)
    # C_lambda = []
    C_lambda = np.zeros((image.shape[0], image.shape[1]))
    # calculate matrix C for each (x,y)
    for i in range(0, image.shape[0]):
        for j in range(0, image.shape[1]):
            start_i, end_i = i-windowSize//2, i+windowSize//2
            start_j, end_j = j-windowSize//2, j+windowSize//2
            start_i = 0 if start_i <0 else start_i
            start_j = 0 if start_j <0 else start_j
            end_i = image.shape[0] if end_i > image.shape[0] else end_i
            end_j = image.shape[1] if end_j > image.shape[1] else end_j
```

```
                i_x_patch = image_dx[start_i: end_i+1, start_j: end_j+1]
                i_y_patch = image_dy[start_i: end_i+1, start_j: end_j+1]
                assert(i_x_patch.shape == i_y_patch.shape)

                ix2 = np.sum(i_x_patch * i_x_patch)
                iy2 = np.sum(i_y_patch * i_y_patch)
                ixy = np.sum(i_x_patch * i_y_patch)

                # store min(lambda1, lambda2) for each C(x,y)
                C_x_y = np.array([[ix2, ixy], [ixy, iy2]])
                # C_lambda.append((min(np.linalg.eigvals(C_x_y)), (j, i)))
                C_lambda[i, j] = min(np.linalg.eigvals(C_x_y))


        # non-maximum supression
        C_lambda_supress = []
        for i in range(0, image.shape[0], windowSize):
            for j in range(0, image.shape[1], windowSize):
                start_i, end_i = i-windowSize//2, i+windowSize//2
                start_j, end_j = j-windowSize//2, j+windowSize//2
                start_i = 0 if start_i <0 else start_i
                start_j = 0 if start_j <0 else start_j
                end_i = image.shape[0] if end_i > image.shape[0] else end_i
                end_j = image.shape[1] if end_j > image.shape[1] else end_j

                lambda_patch = C_lambda[start_i: end_i, start_j: end_j]
                max_lambda = max(lambda_patch.flatten())
                for k in range(0, lambda_patch.shape[0]):
                    for l in range(0, lambda_patch.shape[1]):
                        if lambda_patch[k, l] == max_lambda:
                            C_lambda_supress.append((max_lambda, (start_j+l, start_i+k)))
                            continue

        # Find top nCorners
        # C_lambda.sort()
        # C_lambda.reverse()
        C_lambda_supress.sort()
        C_lambda_supress.reverse()
        corners = np.zeros((nCorners, 2), dtype='int32')
        for i in range(0, corners.shape[0]):
            corners[i, :] = C_lambda_supress[i][1]
            # corners[i, :] = C_lambda[i][1]

        return corners

In [4]: def show_corners_result(imgs, corners):
        fig = plt.figure(figsize=(8, 8))
```

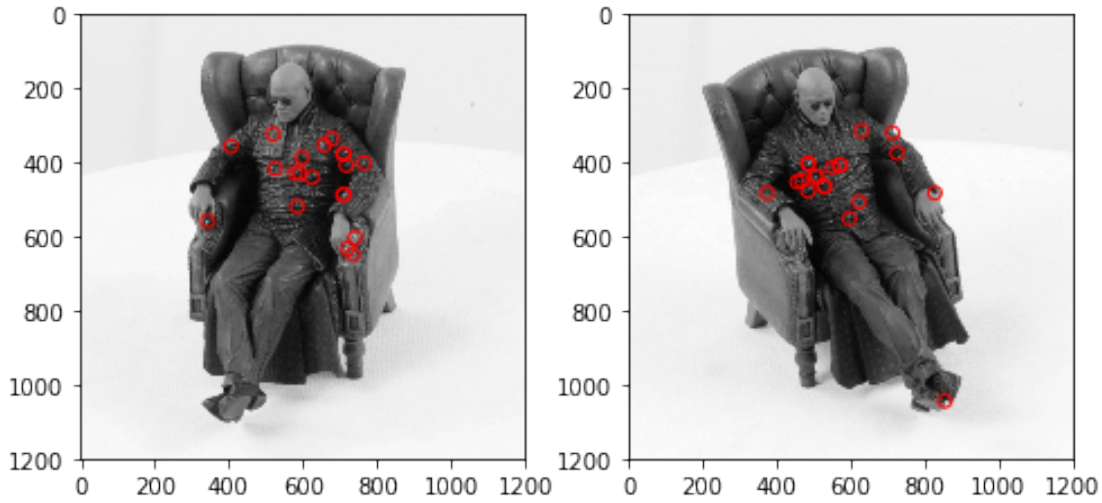
```
            ax1 = fig.add_subplot(221)
            ax1.imshow(imgs[0], cmap='gray')
            ax1.scatter(
                corners[0][:, 0], corners[0][:, 1], s=35, edgecolors='r', facecolors='none')

            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()
```

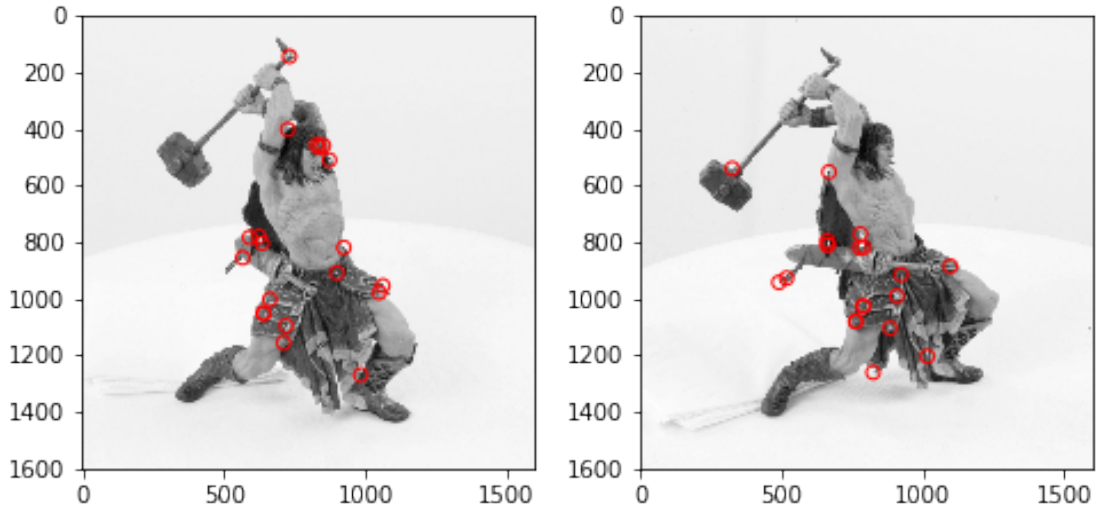
In [5]:
```
# detect corners on warrior and matrix sets
# adjust your corner detection parameters here
nCorners = 20

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


for idx in range(0, len(windowSize)):
    # 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')

        #img_mat = imresize(img_mat, size=0.5)
        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[idx], windowSize

        img_war = imread('p4/warrior/warrior' + str(i) + '.png')
        #img_war = imresize(img_war, size=0.5)
        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[idx], windowSize

    print("Corners detect with smoothSTD:{}, windowSize:{}".format(smoothSTD[idx], win
    show_corners_result(imgs_mat, crns_mat)
    show_corners_result(imgs_war, crns_war)
```

```
/home/amanraj/anaconda3/envs/neural/lib/python3.6/site-packages/ipykernel_launcher.py:16: Depre
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.
```

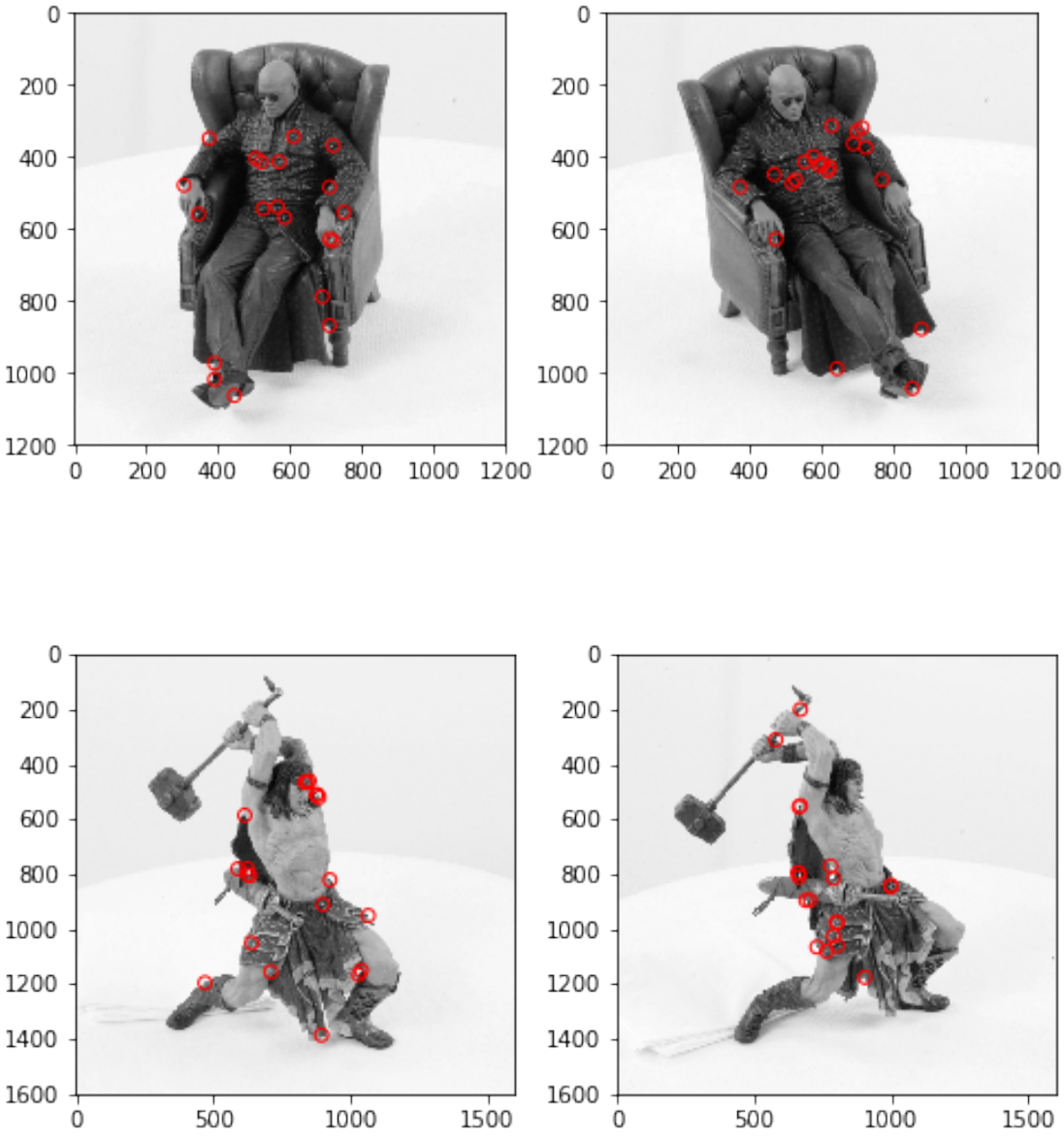
```
  app.launch_new_instance()
/home/amanraj/anaconda3/envs/neural/lib/python3.6/site-packages/ipykernel_launcher.py:24: Depre
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.
```

Corners detect with smoothSTD:0.5, windowSize:3

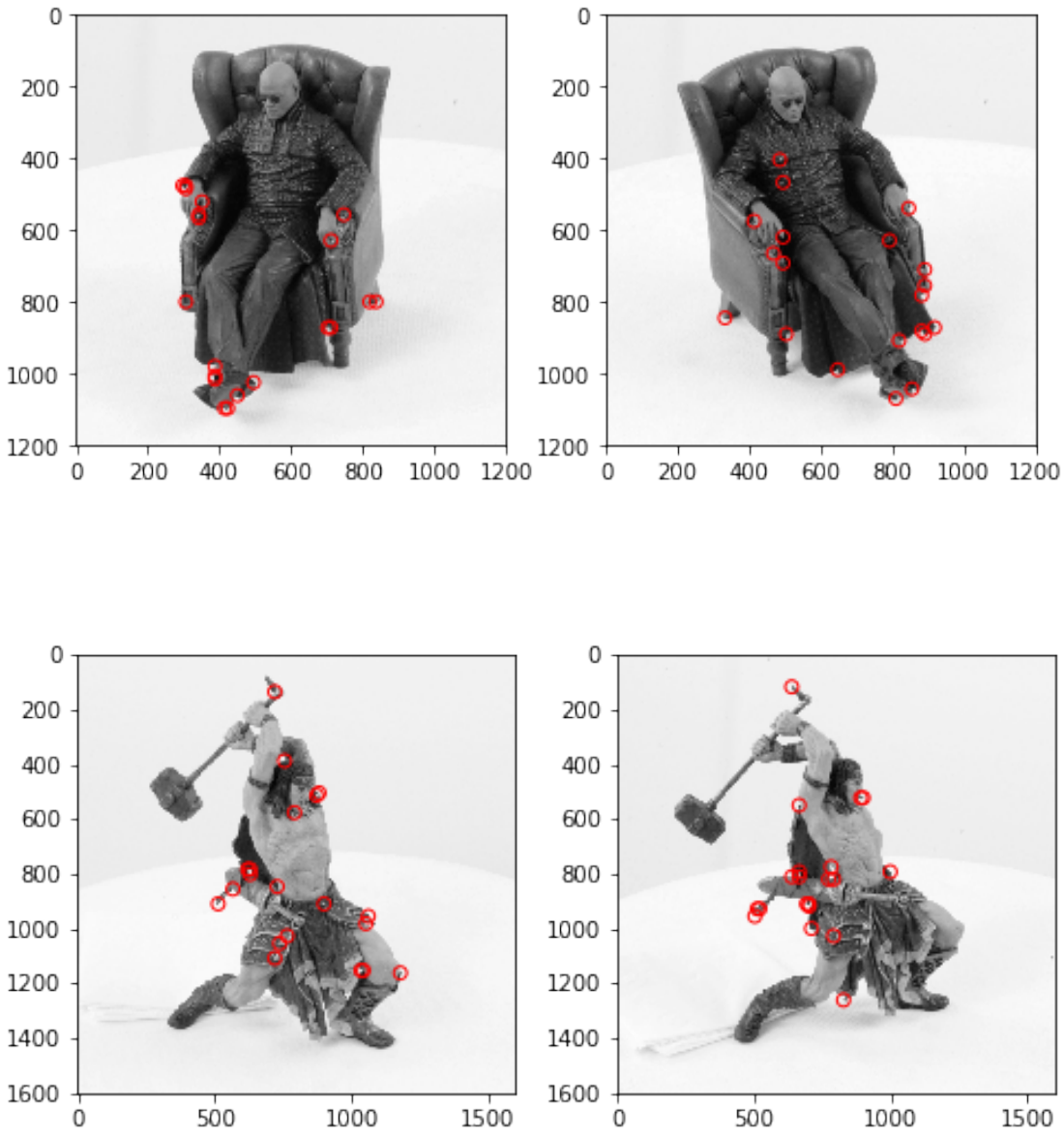




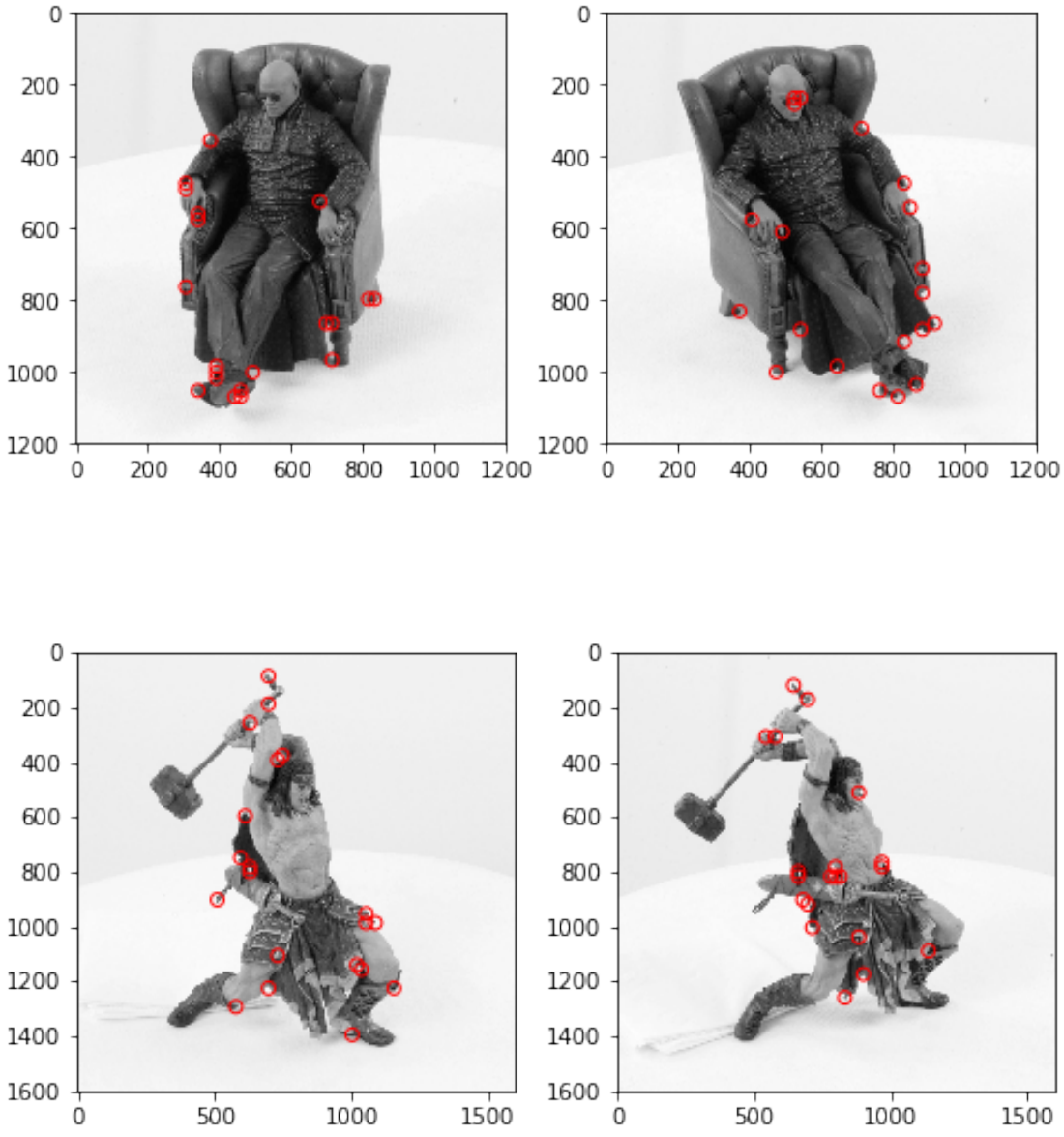
Corners detect with smoothSTD:1, windowSize:5



Corners detect with smoothSTD:2, windowSize:9



Corners detect with smoothSTD:4, windowSize:17



When we increase the standard deviation, the smoothening operation makes image better as it removes noise but with higher values it also start effecting edges, and corners. Therefore, the corners that were detected in the middle of the image is no longer visible with higher value of stddev. When we increase window size, the corner points are more scattered as the non-max suppression is happening over a larger area.

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

Write a function ncc_match that implements the NCC matching algorithm for two input windows. NCC = $\sum_{i,j} \tilde{W}_1(i,j) \cdot \tilde{W}_2(i,j)$ where $\tilde{W} = \frac{W - \overline{W}}{\sqrt{\sum_{k,l}(W(k,l) - \overline{W})^2}}$ is a mean-shifted and normalized version of the window and $\overline{W}$ is the mean pixel value in the window W.

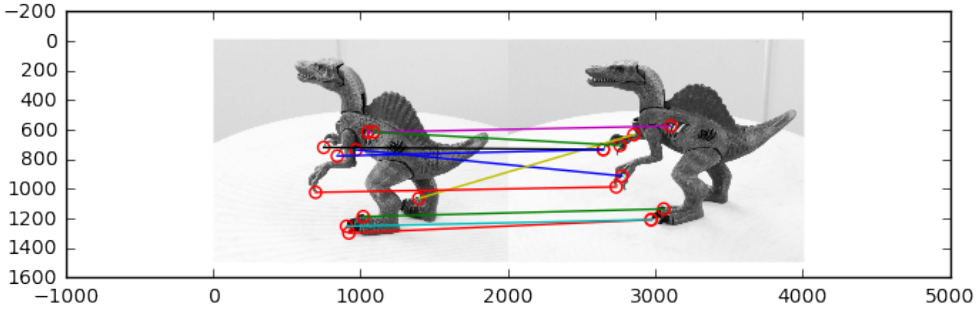
In [6]:
```
import math
def ncc_match(img1, img2, c1, c2, R):
    """Compute NCC given two windows.

    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.
    """
    def normalize_window(W):
        mean = np.mean(W)
        W = W - mean
        W = W / math.sqrt(np.sum(W*W))
        return W

    windowS = 2 * R + 1
    W1 = img1[c1[1]-windowS//2: c1[1]+windowS//2+1, \
              c1[0]-windowS//2: c1[0]+windowS//2+1]
    W2 = img2[c2[1]-windowS//2: c2[1]+windowS//2+1, \
              c2[0]-windowS//2: c2[0]+windowS//2+1]
    W1 = normalize_window(W1)
    W2 = normalize_window(W2)

    matching_score = np.sum(W1 * W2)
    return matching_score
```

In [7]:
```
# 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
```

1.5.3 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 [8]: 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.

            """
            matching = []
            for i in range(corners1.shape[0]):
                corn1 = corners1[i]
                score_max = None
                match_pair = None
                for j in range(corners2.shape[0]):
                    corn2 = corners2[i]
                    score = ncc_match(img1, img2, corn1, corn2, R)
                    if (score_max is None) or (score > score_max):
                        score_max = score
                        match_pair = (corn1, corn2)
```

```
                if score_max >= NCCth:
                    matching.append(match_pair)

            return matching
```

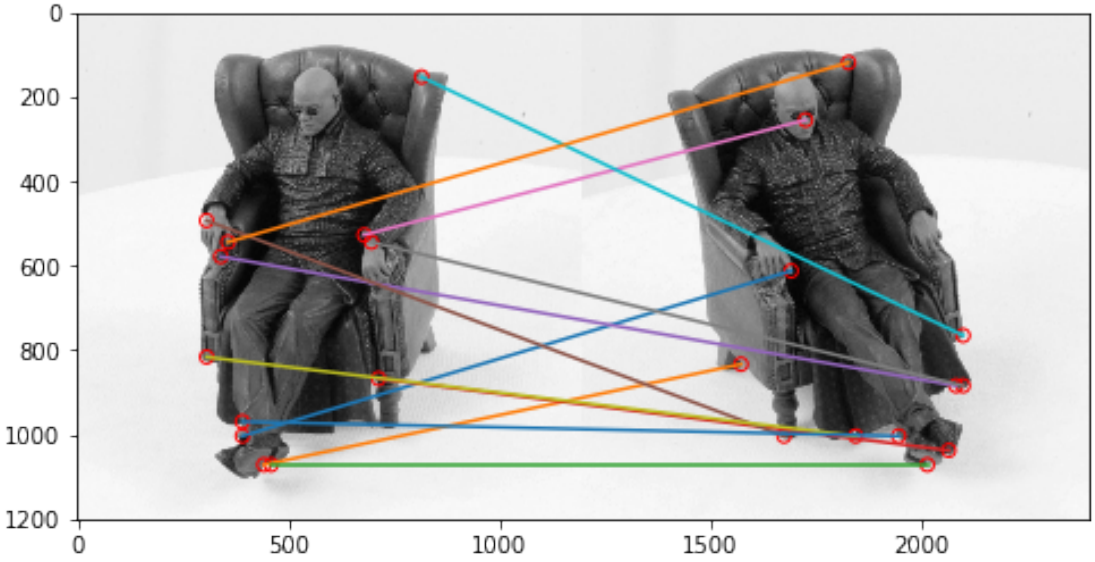
In [9]:
```
# detect corners on warrior and matrix sets
# adjust your corner detection parameters here
nCorners = 30
smoothSTD = 4
windowSize = 17

# 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, windowSize))

    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, windowSize))
```

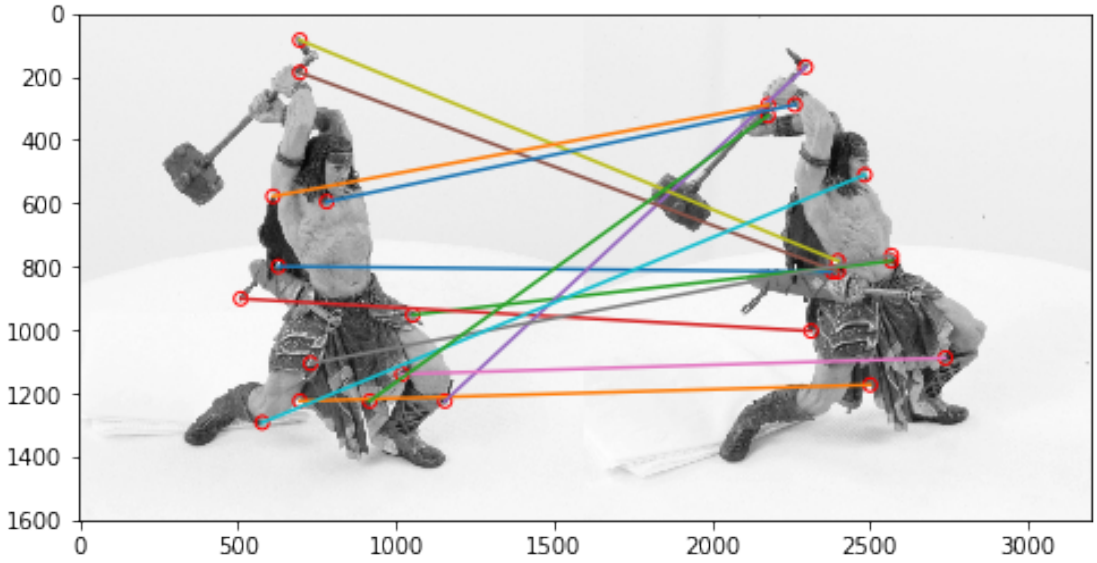
```
/home/amanraj/anaconda3/envs/neural/lib/python3.6/site-packages/ipykernel_launcher.py:13: Depre
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.
  del sys.path[0]
/home/amanraj/anaconda3/envs/neural/lib/python3.6/site-packages/ipykernel_launcher.py:19: Depre
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.
```

In [10]:
```
#show_corners_result(imgs_mat, crns_mat)
#show_corners_result(imgs_war, crns_war)
```

In [11]:
```
# match corners
R = 15
NCCth_mat = 0.02
NCCth_war = 0.2
matching_mat = naive_matching(
    imgs_mat[0]/255, imgs_mat[1]/255, crns_mat[0], crns_mat[1], R, NCCth_mat)
matching_war = naive_matching(
    imgs_war[0]/255, imgs_war[1]/255, crns_war[0], crns_war[1], R, NCCth_war)
```

In [12]:
```
# plot matching result
def show_matching_result(img1, img2, matching):
    fig = plt.figure(figsize=(8, 8))
    plt.imshow(np.hstack((img1, img2)), cmap='gray')
    for p1, p2 in matching:
        plt.scatter(p1[0], p1[1], s=35, edgecolors='r', facecolors='none')
        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)
```

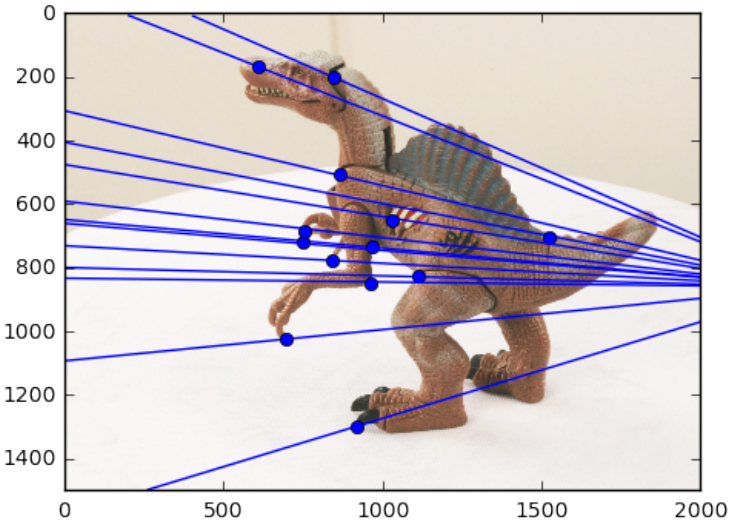


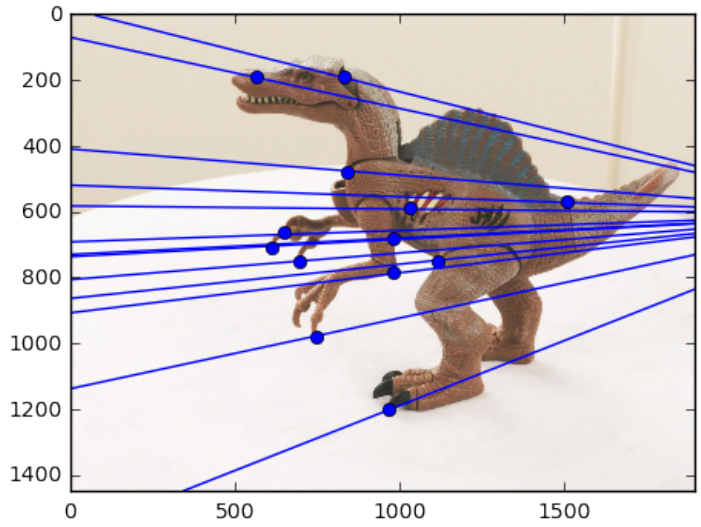


1.5.4 Epipolar Geometry [4 pts]

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 [13]: 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
    and corner point correspondences input:
    F--> Fundamental matrix
    output:
    e1--> corresponding epipole in image 1
    e2--> epipole in image2
    '''
    # caclulate epipole e1
    e1 = np.linalg.svd(F.T)[-1]
    # convert into cartesian coordinates
    e1 = e1[-1, :]/e1[-1, :][-1]
    # caclulate epipole e2
    e2 = np.linalg.svd(F)[-1]
    # convert into cartesian coordinates
    e2 = e2[-1, :]/e2[-1, :][-1]
```

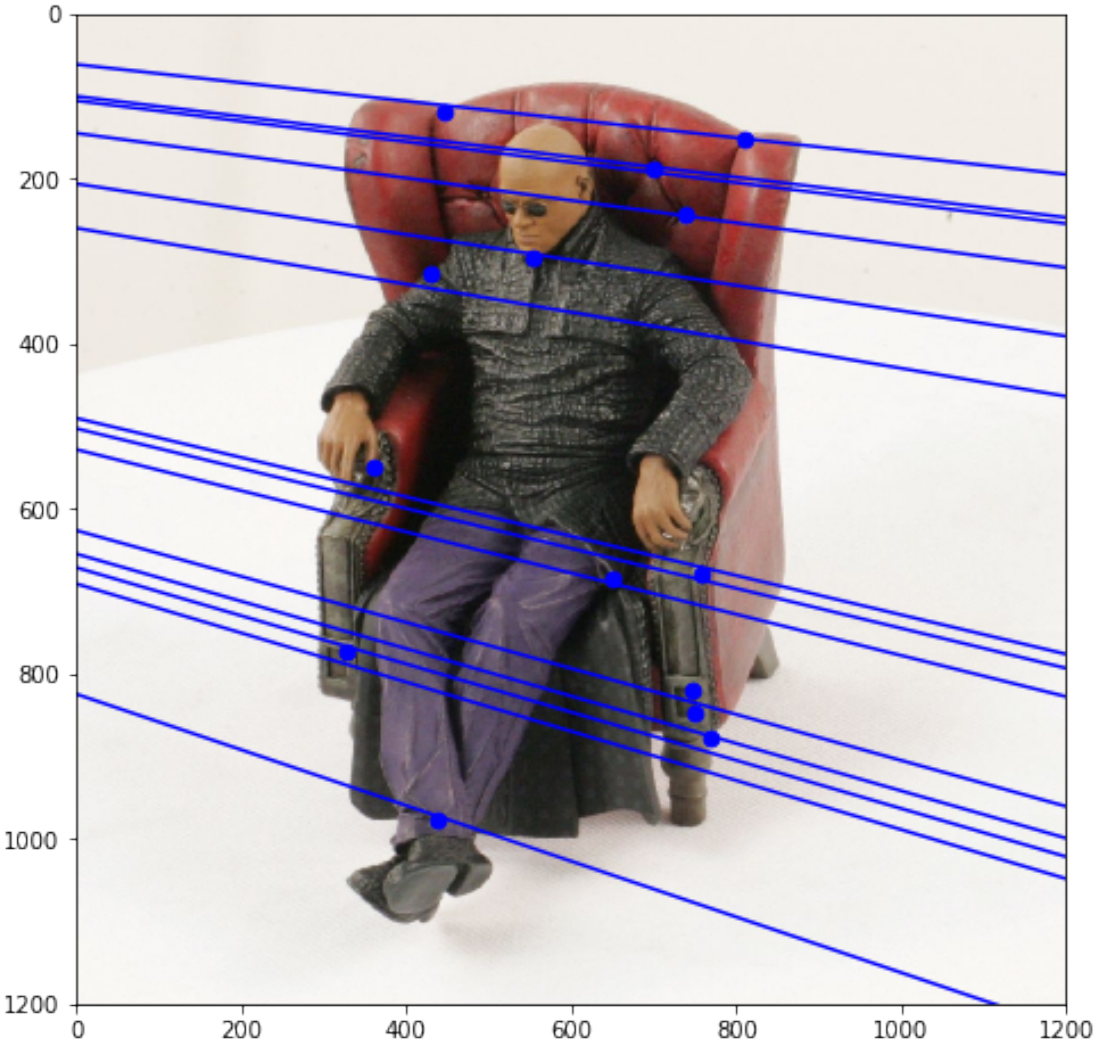
```
            return e1,e2
```

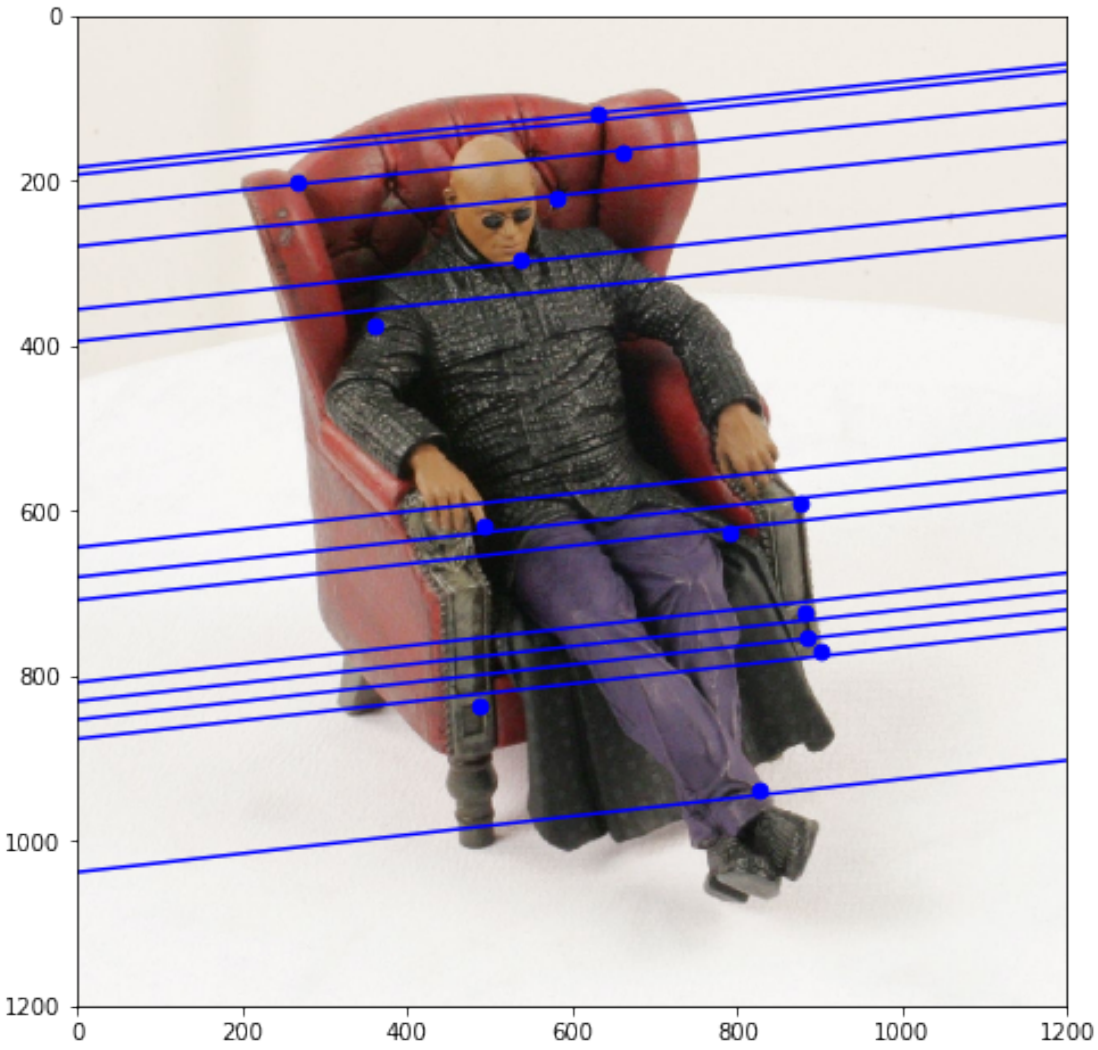
In [14]:
```
def plot_epipolar_lines(img1, img2, cor1, cor2):
    """Plot epipolar lines on image given image, corners

    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)

    """
    # Find fundamental matrix
    F = fundamental_matrix(cor1, cor2)

    fig1 = plt.figure(figsize=(8, 8))
    plt.imshow(img1, cmap = 'gray')
    for i in range(cor1.shape[1]):
        # Since given homogeneous points have last coordinate=1
        # So we can directly plot them using first two coordinates
        plt.scatter(cor1[0][i], cor1[1][i], s=45, edgecolors='b', facecolors='b')
    for i in range(cor2.shape[1]):
        a_img1 = np.dot(F, cor2[:,i])
        x1, x2 = 0, img1.shape[1]
        # Find points (x1,y1) and (x2,y2) on the epipolar line
        # Using the equation Ax+By+C = 0
        y1 = a_img1[2]/(-a_img1[1])
        y2 = (x2*a_img1[0]+a_img1[2])/(-a_img1[1])
        plt.plot([x1, x2], [y1, y2], color = 'b')
        plt.axis([0,img1.shape[1],img1.shape[0],0])

    plt.show()

    fig2 = plt.figure(figsize=(8, 8))
    plt.imshow(img2, cmap = 'gray')
    for i in range(cor2.shape[1]):
        # Since given homogeneous points have last coordinate=1
        # So we can directly plot them using first two coordinates
        plt.scatter(cor2[0][i], cor2[1][i], s=45, edgecolors='b', facecolors='b')
    for i in range(cor1.shape[1]):
        b_img2 = np.dot(F.T, cor1[:,i])
        x1, x2 = 0, img2.shape[1]
        # Find points (x1,y1) and (x2,y2) on the epipolar line
        # Using the equation Ax+By+C = 0
        y1 = b_img2[2]/(-b_img2[1])
        y2 = (x2*b_img2[0]+b_img2[2])/(-b_img2[1])
        plt.plot([x1, x2], [y1, y2], color = 'b')
        plt.axis([0,img2.shape[1],img2.shape[0],0])
```

```
            plt.show()
```

In [15]:
```
# replace images and corners with those of matrix and warrior
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")

plot_epipolar_lines(I1,I2,cor1,cor2)
```

```
/home/amanraj/anaconda3/envs/neural/lib/python3.6/site-packages/ipykernel_launcher.py:2: Depre
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.

/home/amanraj/anaconda3/envs/neural/lib/python3.6/site-packages/ipykernel_launcher.py:3: Depre
`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 imports until
```

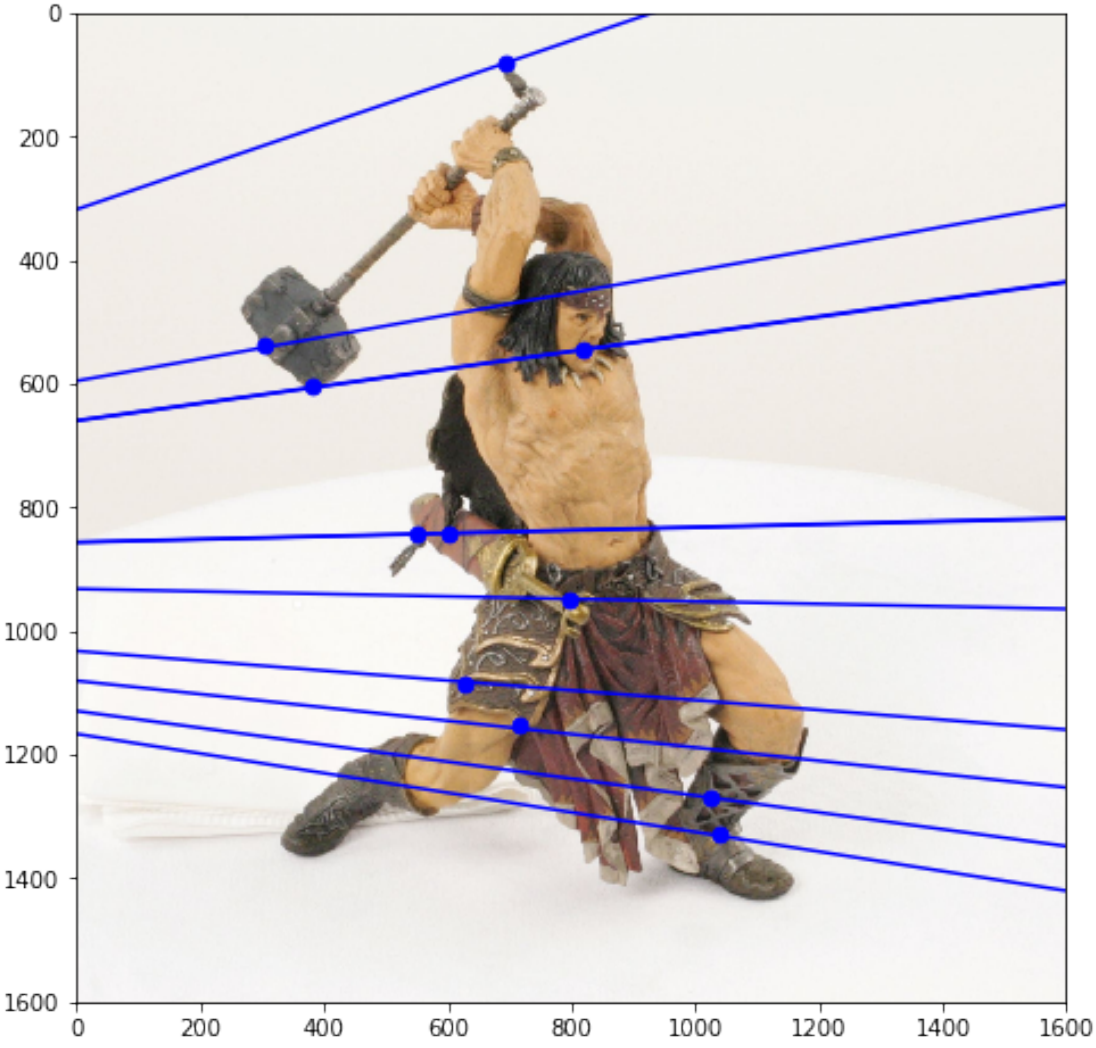

0
200
400
600
800
1000
1200
0
200
400
600
800
1000
1200

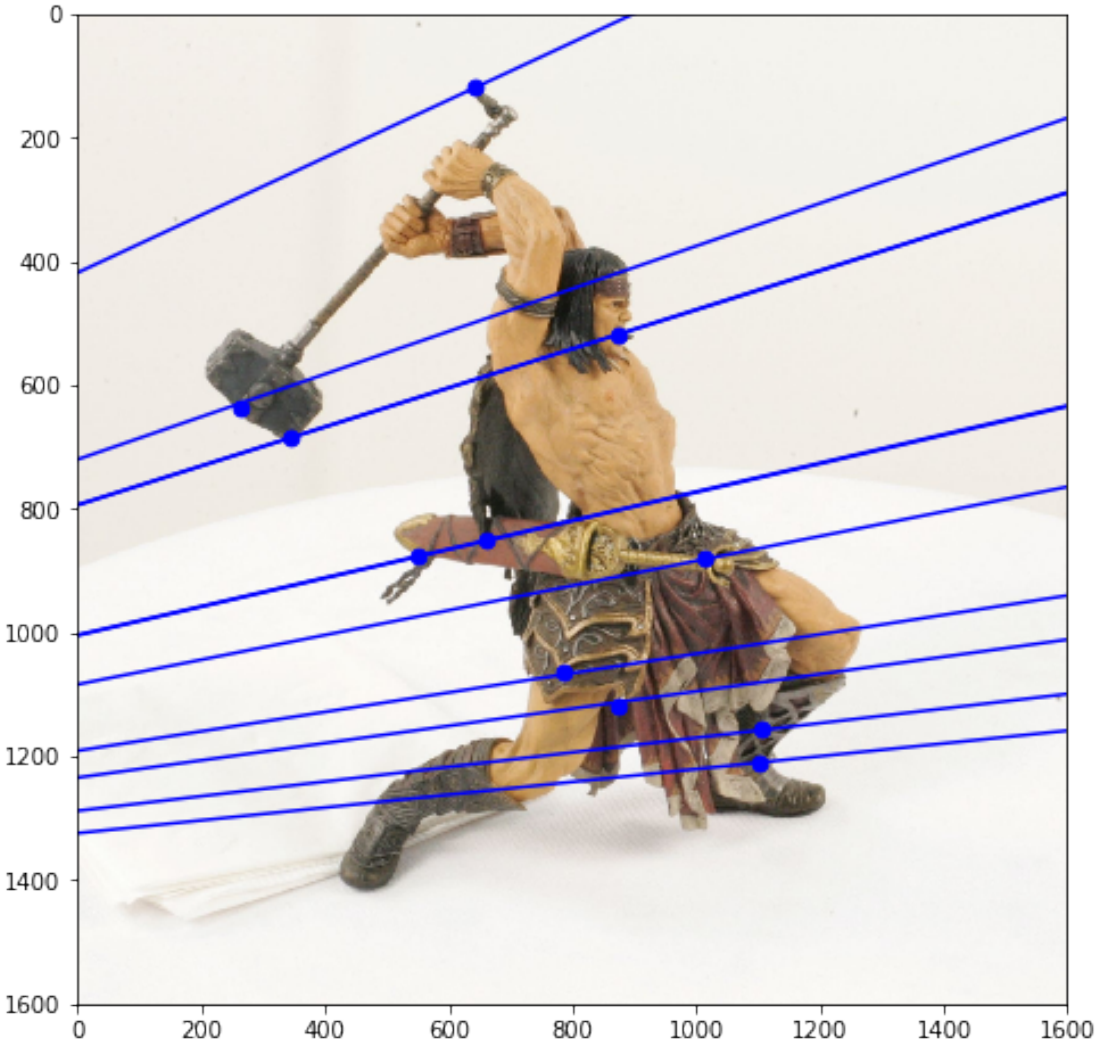


In [16]:
```
# replace images and corners with those of matrix and warrior
I1 = imread("./p4/warrior/warrior0.png")
I2 = imread("./p4/warrior/warrior1.png")

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

plot_epipolar_lines(I1,I2,cor1,cor2)
```

```
/home/amanraj/anaconda3/envs/neural/lib/python3.6/site-packages/ipykernel_launcher.py:2: Deprec
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.

/home/amanraj/anaconda3/envs/neural/lib/python3.6/site-packages/ipykernel_launcher.py:3: Deprec
`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 imports until
```

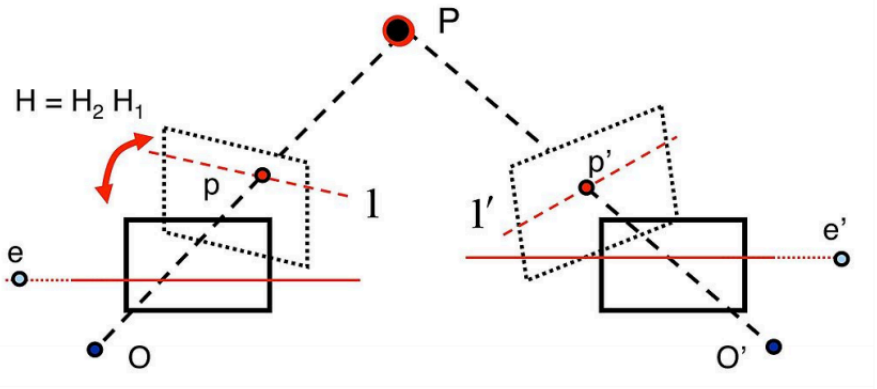




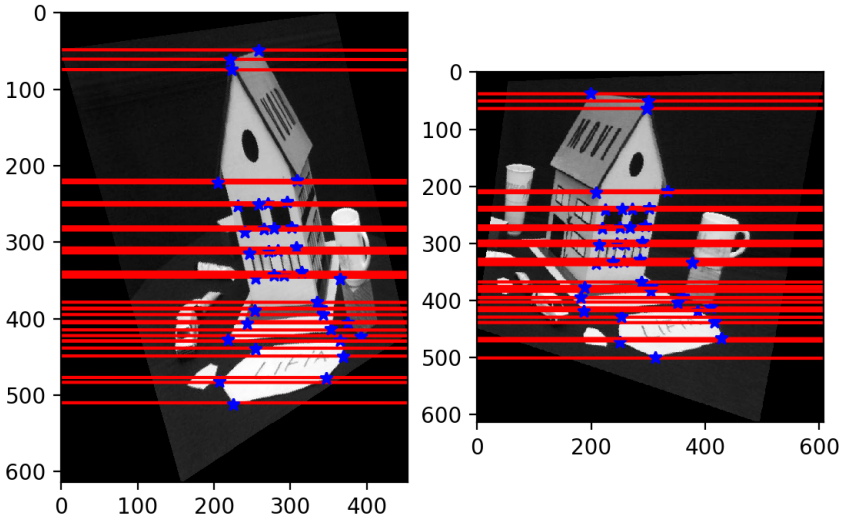
1.5.5 Image Rectification [3 pts]

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 $\mathtt{E} = [\boldsymbol{T}_x]\boldsymbol{R} = [\boldsymbol{T}_x]$. 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:




```
In [17]: def compute_matching_homographies(e2, F, im2, points1, points2):

             '''This function computes the homographies to get the rectified images
             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
```

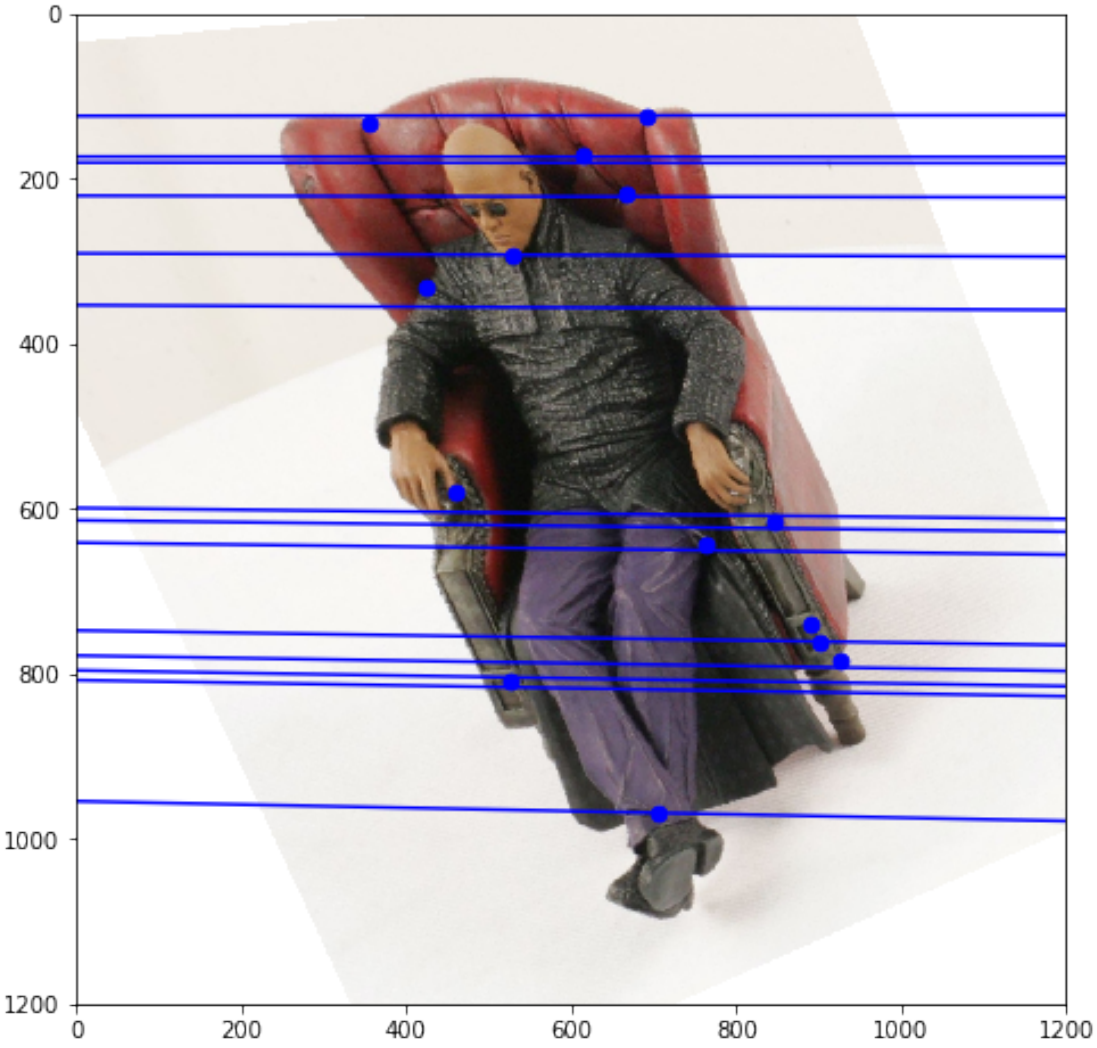
```
    W = points1_hat / points1_hat[:, 2].reshape(-1, 1)
    b = (points2_hat / points2_hat[:, 2].reshape(-1, 1))[:, 0]

    # 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

# convert points from euclidian to homogeneous
def to_homog(points):
    ones = [1. for _ in range(points.shape[1])]
    # append 1. for all points as last coordinate
    points = np.append(points, [ones], axis=0)

    return points


# convert points from homogeneous to euclidian
def from_homog(points_homog):
    # get all the last element for all homogeneous points
    w = points_homog[-1:][0]
    points_homog = np.delete(points_homog, -1, 0)

    # now divide each point with it's corresponding w
    for idx, wi in enumerate(w):
        points_homog[:, idx] /= wi

    return points_homog

def rectify_image(H, source_image):
    target_image = np.ones(source_image.shape)

    for x in range(0, target_image.shape[1]):
        for y in range(0, target_image.shape[0]):
            # convert to homogeneous coordinate
            xy_homo = to_homog(np.array([[x, y]]).T)
            # apply homography
            mapped_x, mapped_y = from_homog(np.dot(H, xy_homo)).T[0]
            if 0<=int(mapped_y)<target_image.shape[0] and \
             0<=int(mapped_x)<target_image.shape[1]:
                target_image[y, x, :] = source_image[int(mapped_y), int(mapped_x), :]

    return target_image

def rectify_points(H, points):
```

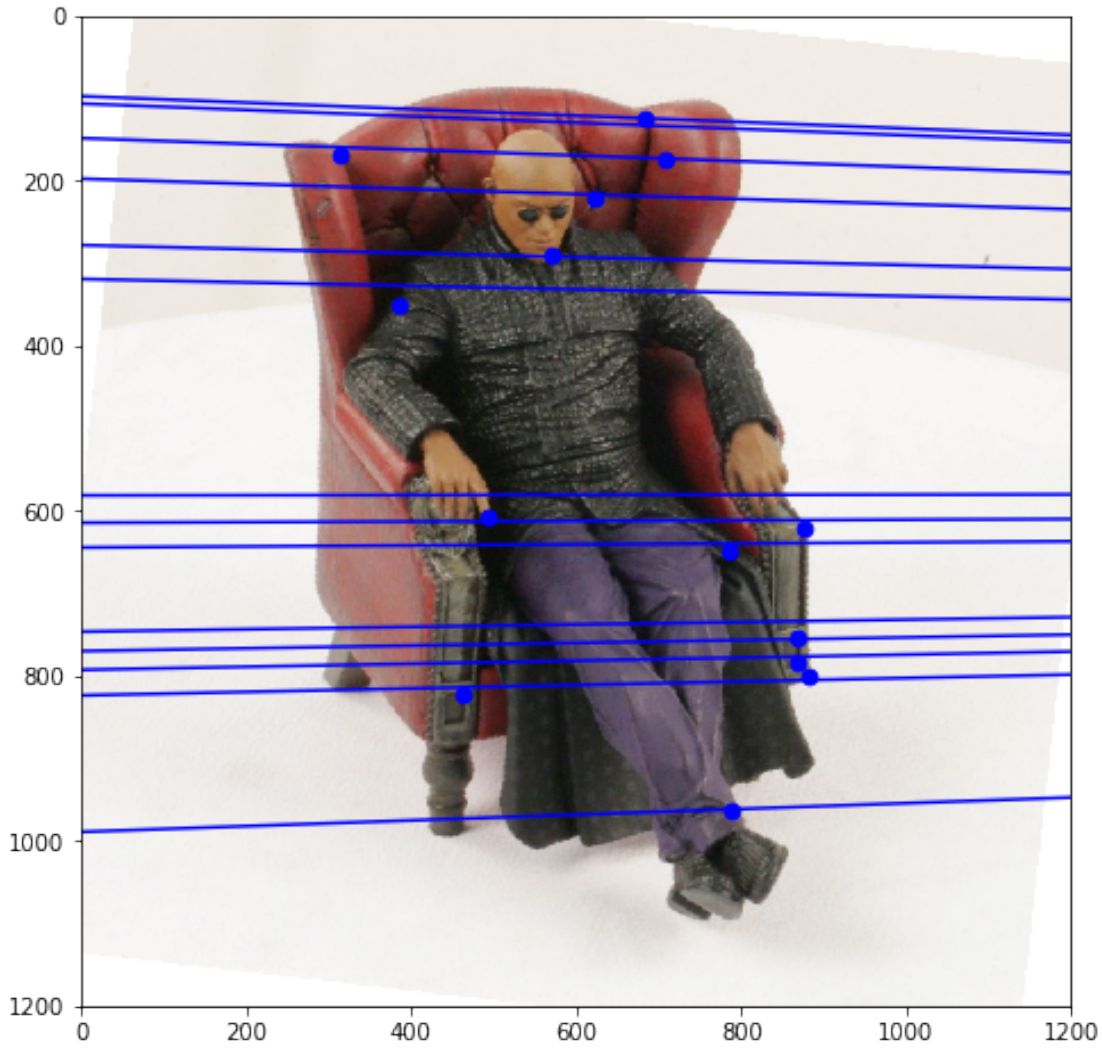
```
            for i in range(points.shape[1]):
                points[:, i] = np.dot(H, points[:, i])
                # convert the points to cartesian
                points[:, i] = points[:, i]/points[:, i][-1]

            return points

        def image_rectification(im1,im2,points1,points2):
            '''this function provides the rectified images along with the new
            corner 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
            outpu:
            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
            '''
            # Compute fundamental matrix
            F = fundamental_matrix(points1, points2)
            # Compute epipoles
            e1, e2 = compute_epipole(F)
            # Compute homography
            H1, H2 = compute_matching_homographies(e2, F.T, im2, points1.T, points2.T)
            # rectify images
            rectified_im1 = rectify_image(np.linalg.inv(H1), im1/np.max(im1))
            rectified_im2 = rectify_image(np.linalg.inv(H2), im2/np.max(im2))
            # rectify points
            new_cor1 = rectify_points(H1, points1)
            new_cor2 = rectify_points(H2, points2)

            return rectified_im1, rectified_im2, new_cor1, new_cor2
```

In [18]:
```
# replace images and corners with those of matrix and warrior
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")

#image_rectification(I1, I2, cor1, cor2)
I1, I2, cor1, cor2 = image_rectification(I1, I2, cor1, cor2)
plot_epipolar_lines(I1, I2, cor1, cor2)
```

```
/home/amanraj/anaconda3/envs/neural/lib/python3.6/site-packages/ipykernel_launcher.py:2: Depre
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
```

```
Use ``imageio.imread`` instead.

/home/amanraj/anaconda3/envs/neural/lib/python3.6/site-packages/ipykernel_launcher.py:3: Deprec
`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 imports until
/home/amanraj/anaconda3/envs/neural/lib/python3.6/site-packages/ipykernel_launcher.py:61: Futu
To use the future default and silence this warning we advise to pass `rcond=None`, to keep usi
```

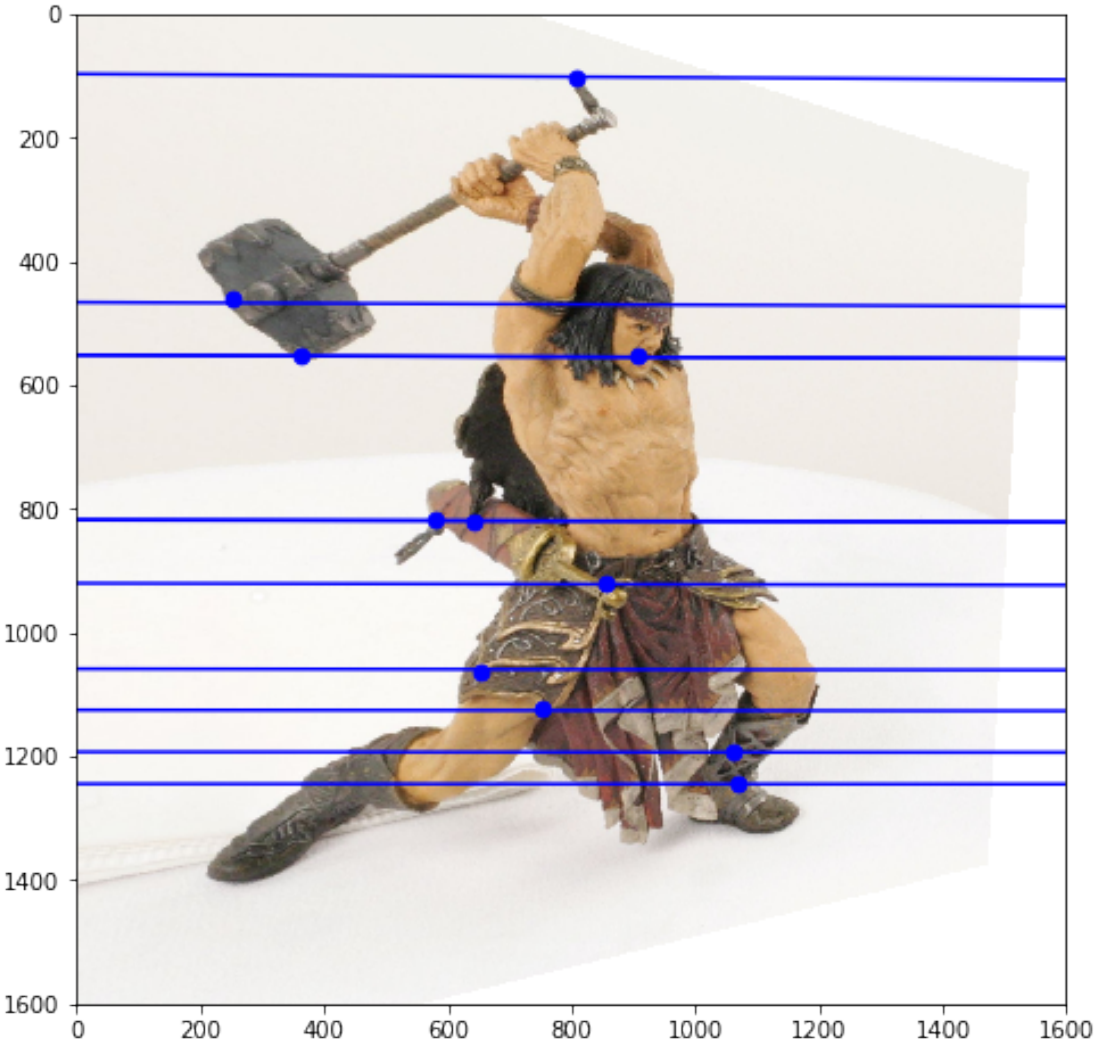


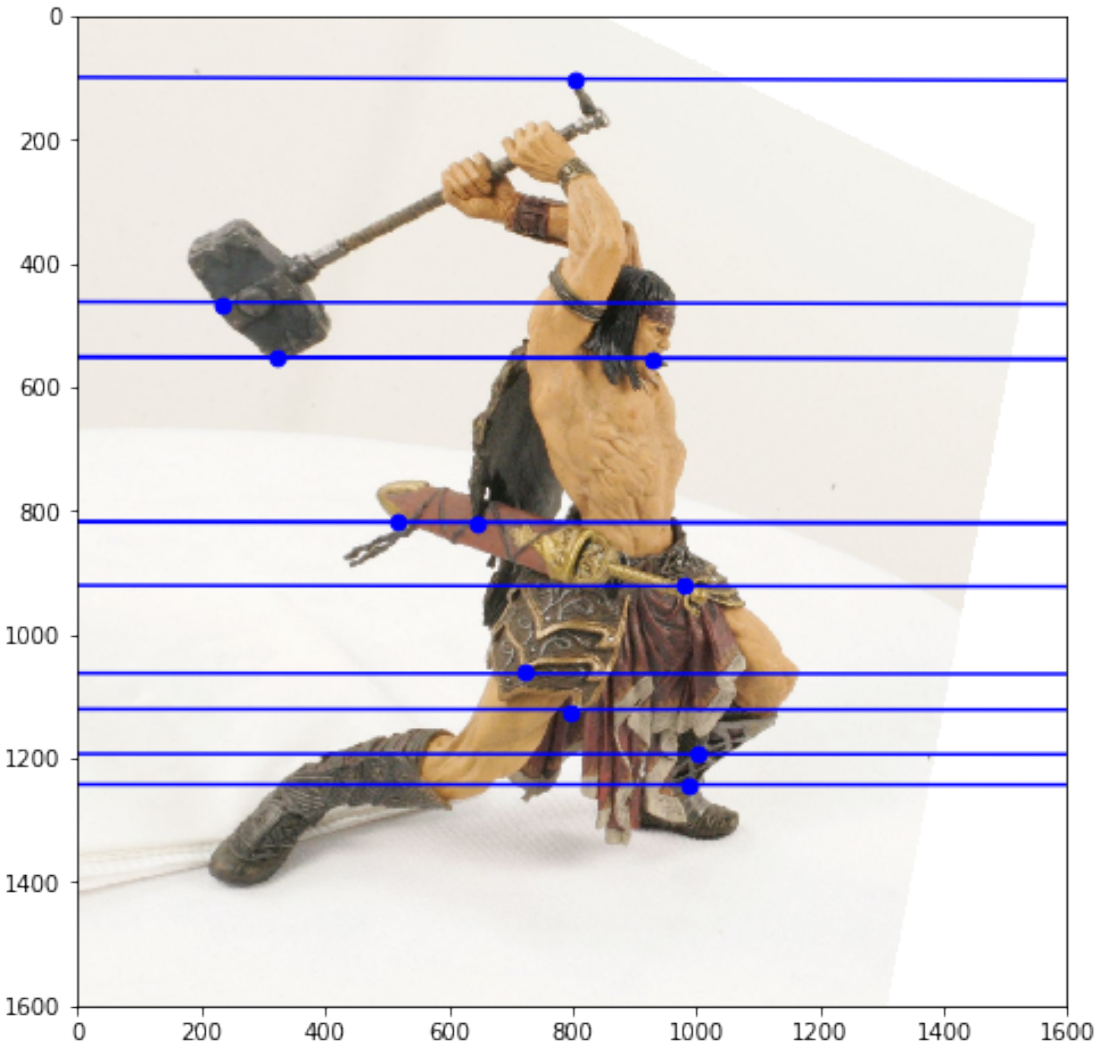


In [19]:
```
# replace images and corners with those of matrix and warrior
I1 = imread("./p4/warrior/warrior0.png")
I2 = imread("./p4/warrior/warrior1.png")

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

I1, I2, cor1, cor2 = image_rectification(I1, I2, cor1, cor2)
plot_epipolar_lines(I1,I2,cor1,cor2)
```

```
/home/amanraj/anaconda3/envs/neural/lib/python3.6/site-packages/ipykernel_launcher.py:2: Deprec
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.

/home/amanraj/anaconda3/envs/neural/lib/python3.6/site-packages/ipykernel_launcher.py:3: Deprec
```

```
`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 imports until
/home/amanraj/anaconda3/envs/neural/lib/python3.6/site-packages/ipykernel_launcher.py:61: Futu
To use the future default and silence this warning we advise to pass `rcond=None`, to keep usi
```





1.5.6 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.

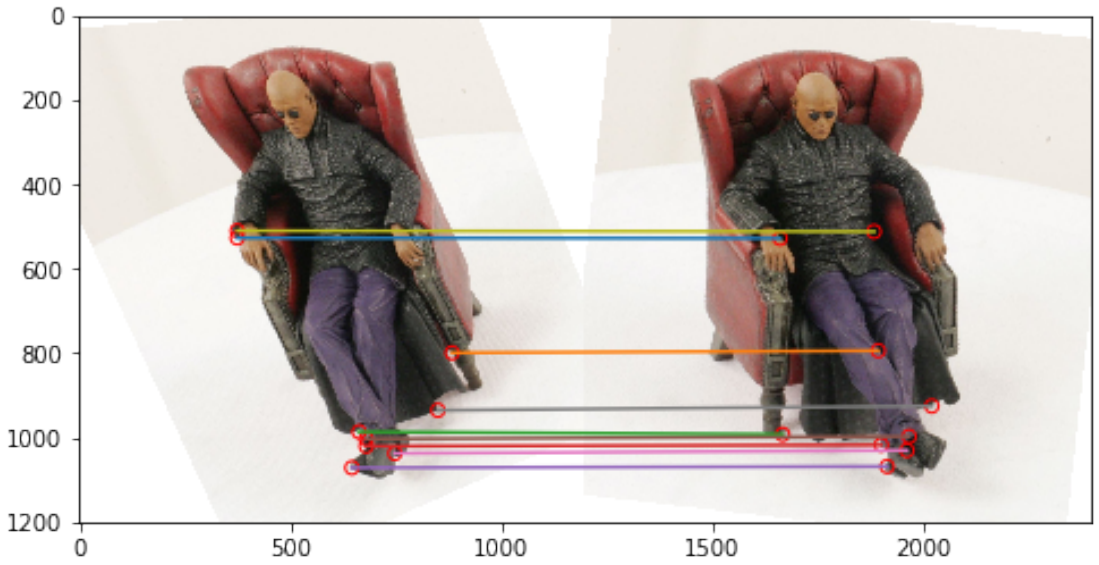
```
In [20]: def display_correspondence(img1, img2, corrs, name):
             """Plot matching result on image pair given images and correspondences

             Args:
                 img1: Image 1.
```

```
        img2: Image 2.
        corrs: Corner correspondence

    """
    fig = plt.figure(figsize=(8, 8))
    plt.imshow(np.hstack((img1, img2)), cmap='gray')
    for p1, p2 in corrs:
        plt.scatter(p1[0], p1[1], s=35, edgecolors='r', facecolors='none')
        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(name + '_matching.png')
    plt.show()


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

    """
    matching = []
    corners1 = to_homog(corners1.T)
    for i in range(0, corners1.shape[1]):
        a = np.dot(F.T, corners1[:, i])
        score_max = None
        match_pair = None
        # we will use line equation Ax+By+C = 0
        for x in range(R, img2.shape[1]-R):
            y = int((a[0]*x + a[2])/(-a[1]))
            # skiped for outside window size
            if y < R or y > img2.shape[0]-R:
                continue
            corn1 = corners1[:2, i].astype(np.int32)
            corn2 = np.array([x, y]).astype(np.int32)
            # calculate ncc match similarity
            score = ncc_match(img1, img2, corn1, corn2, R)
```

```
                    if (score_max is None) or (score > score_max):
                        score_max = score
                        match_pair = (corn1, corn2)

                if score_max > NCCth:
                    matching.append(match_pair)

        return matching
```

In [21]:
```
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")

rectified_im1,rectified_im2,new_cor1,new_cor2 = image_rectification(I1,I2,cor1,cor2)
F_new = fundamental_matrix(new_cor1, new_cor2)

nCorners = 10
#decide the NCC matching window radius
R = 10
NCCth = 0.6
# detect corners using corner detector here, store in corners1
corners1 = corner_detect(rgb2gray(rectified_im1), nCorners, smoothSTD, windowSize)
corrs = correspondence_matching_epipole(
    rectified_im1, rectified_im2, corners1, F_new, R, NCCth)
display_correspondence(rectified_im1, rectified_im2, corrs, name="matrix")
```

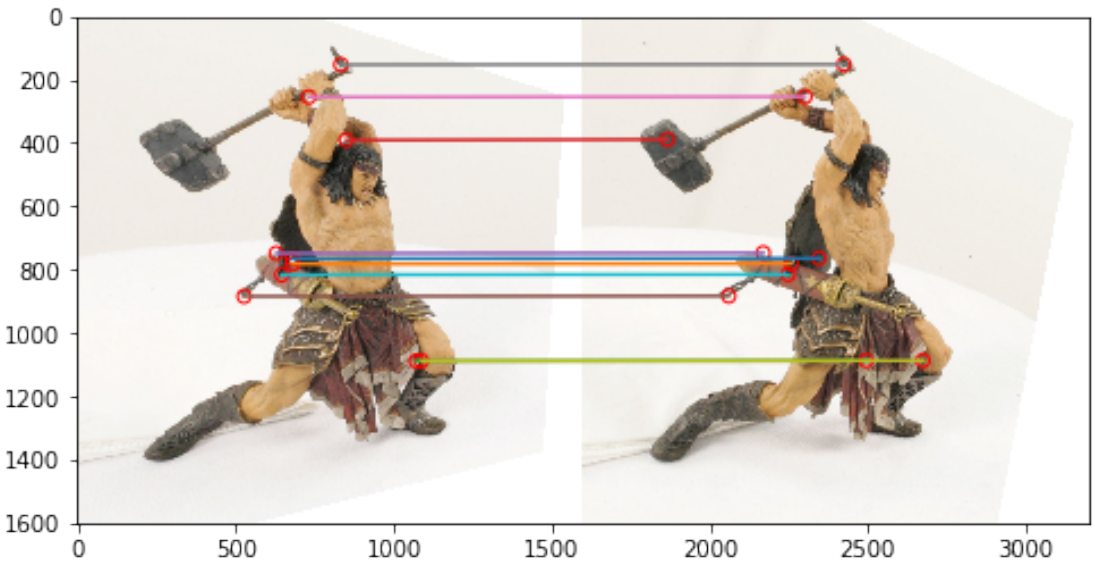
```
/home/amanraj/anaconda3/envs/neural/lib/python3.6/site-packages/ipykernel_launcher.py:1: Depre
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.
  """Entry point for launching an IPython kernel.
/home/amanraj/anaconda3/envs/neural/lib/python3.6/site-packages/ipykernel_launcher.py:2: Depre
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.

/home/amanraj/anaconda3/envs/neural/lib/python3.6/site-packages/ipykernel_launcher.py:61: Futu
To use the future default and silence this warning we advise to pass `rcond=None`, to keep usi
/home/amanraj/anaconda3/envs/neural/lib/python3.6/site-packages/ipykernel_launcher.py:18: Runt
```




```
In [22]: 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")

         rectified_im3,rectified_im4,new_cor3,new_cor4 = image_rectification(I3,I4,cor3,cor4)
         F_new2=fundamental_matrix(new_cor3, new_cor4)

         nCorners = 10
         #decide the NCC matching window radius
         NCCth = 0.6
         R = 10

         corners2 = corner_detect(rgb2gray(rectified_im3), nCorners, smoothSTD, windowSize)
         corrs = correspondence_matching_epipole(
             rectified_im3, rectified_im4, corners2, F_new2, R, NCCth)
         display_correspondence(rectified_im3, rectified_im4, corrs, name="warrior")
```

```
/home/amanraj/anaconda3/envs/neural/lib/python3.6/site-packages/ipykernel_launcher.py:1: Depre
`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.
/home/amanraj/anaconda3/envs/neural/lib/python3.6/site-packages/ipykernel_launcher.py:2: Depre
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.

/home/amanraj/anaconda3/envs/neural/lib/python3.6/site-packages/ipykernel_launcher.py:61: Futu
To use the future default and silence this warning we advise to pass `rcond=None`, to keep usi
```

```
/home/amanraj/anaconda3/envs/neural/lib/python3.6/site-packages/ipykernel_launcher.py:18: Runti
```



Q.2.

(a) $t' = [t_x, 0, 0]^T$, $R = I$

$\therefore \quad E = [t'_x] R$ where t'_x is a skew symmetric matrix

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

① Epipoles:

$E e_B = 0 \Rightarrow e_A = e_B = [1, 0, 0]^T \quad [\because t_y = t_z = 0]$

$E^T e_A = 0$ solving of e_A and e_B

② Epipolar line:

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} = E^T \begin{bmatrix} x_A \\ y_A \\ 1 \end{bmatrix}$$ using homogeneous form of point (x_A, y_A)

$$\therefore \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & +t_x \\ 0 & -t_x & 0 \end{bmatrix} \begin{bmatrix} x_A \\ y_A \\ 1 \end{bmatrix}$$

$$\therefore \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 0 \\ t_x \\ -t_x y_A \end{bmatrix}$$

Using line eq^n: $ax + by + c = 0$

$t_x \cdot y - t_x y_A = 0 \Rightarrow y = y_A$

Hence, $\boxed{y = y_A}$ eq^n of epipolar line l_B in I_B.

(3) Let's find H_A for image I_A which is given by

$$H_A = \begin{bmatrix} e_1^T \\ e_2^T \\ e_3^T \end{bmatrix} \quad \text{where} \quad e_1 = e_A = [1, 0, 0]^T$$

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

$e_3 = e_1 \times e_2 = [0, 0, 1]^T$, putting values we get!

$$H_A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Since we know that $H_B = R H_A$

H_A, H_B homographies

R: Rotation matrix

$\therefore$ for given H_A, there exist H_B; Since $R = I$, given

$$\therefore \quad H_B = R H_A = H_A \Rightarrow \boxed{H_A = H_B = I}$$

Hence, epipolar rectification is possible.

Q.2.

(b) $t' = [t_x, t_y, 0]$; $R = I$

$\therefore E = [t'_x]R \Rightarrow E = \begin{bmatrix} 0 & 0 & t_y \\ 0 & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix}$

Epipoles:

$Ee_B = 0$
$E^T e_A = 0$

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

$\therefore [t_z = 0]$

Epipolar line:

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

$\begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} -t_y \\ t_x \\ t_y x_A - t_x y_A \end{bmatrix}$

$\therefore$ Using line eqn $ax + by + c = 0$

$\therefore -t_y \cdot x + t_x \cdot y + (t_y x_A - t_x y_A) = 0$

Eqn of line l_B in Image I_B.

$$e_1 = e_A, \quad e_2 = \frac{1}{\sqrt{e_1x^2 + e_1y^2}} \left[-e_1y, e_1x, 0\right]^T$$

$$\therefore e_2 = \left[\frac{-t_y}{\sqrt{t_x^2 + t_y^2}}, \frac{t_x}{\sqrt{t_x^2 + t_y^2}}, 0 \right]^T$$

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

$$\therefore H_B = R H_A \quad \text{and} \quad H_A = \begin{bmatrix} e_1^T \\ e_2^T \\ e_3^T \end{bmatrix}$$

$$\therefore H_A = \begin{bmatrix} \frac{t_x}{\sqrt{t_x^2 + t_y^2}} & \frac{t_y}{\sqrt{t_x^2 + t_y^2}} & 0 \\ \frac{t_y}{\sqrt{t_x^2 + t_y^2}} & \frac{t_x}{\sqrt{t_x^2 + t_y^2}} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$\therefore H_B = H_A$ Since $R = I$.

Hence, Epipolar rectification possible. Since we can use these Homographies for I_A & I_B.

Q.2.

(C) $\therefore t'_x = [0, 0, t_z]$; $R = I$

$$E = \begin{bmatrix} 0 & -t_z & 0 \\ t_z & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \text{using} \quad E = [t'_x]R$$

Epipoles:

$$Ee_B = 0 \Rightarrow e_A = e_B = [0, 0, 1]^T$$
$$E^T e_A = 0$$

Since $t_x = t_y = 0$, so only last term exist in the vector

Epipolar line:

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

$$\therefore \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} t_z y_A \\ -t_z x_A \\ 0 \end{bmatrix}$$

Using $ax + by + c = 0$ eqn

$$\therefore t_z y_A \cdot x + (-t_z x_A) \cdot y + 0 = 0$$

or; $y_A \cdot x - x_A \cdot y = 0$

eqn of line l_B in Image I_B

$e_1 = e_A = [0, 0, 1]^T$; $e_2 = [0, 1, 0]^T$

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

$H_B = R H_A$ and $H_A = \begin{bmatrix} e_1^T \\ e_2^T \\ e_3^T \end{bmatrix}$

$\therefore H_A = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$

$\therefore H_B = R H_A = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \therefore \underline{R = I}$

Since, H_A, H_B exists So, epipolar rectification possible!

Q.2.

(d) $\therefore t'_x = [0, 0, 0]$ arbitarary 'R'

$\therefore E = [t_x'] R = 0$ $\sin [t_x'] = 0$

Epipoles:

$\therefore E e_B = 0$

$E^T e_A = 0$ $\Rightarrow$ e_A & e_B Can be any vector, not a unique Solution to the eq^{n}s.

Epipolar lines:

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

There is no line l_B in Image I_B

e_1, e_2, e_3, e_4 Can be arbitarary.

for $H_A = \begin{bmatrix} e_1^T \\ e_2^T \\ e_3^T \end{bmatrix}$ $\therefore$ $H_B = R H_A$

Hence, There is no such H_A & H_B exist for homographpy. Can be any random H_A & H_B.

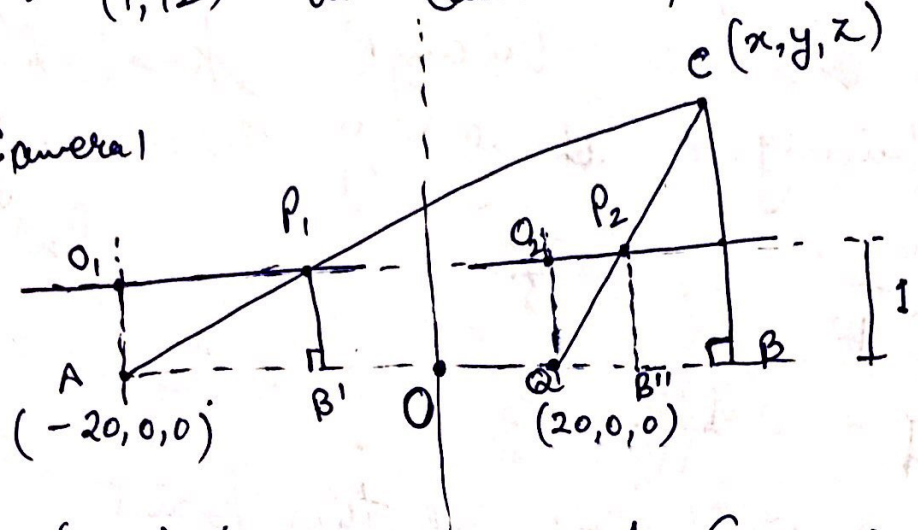
$\therefore$ Epipolar rectification not possible.

Q.1. $P_1' = (12, 12)$ in Camera 1 frame
$P_2' = (1, 12)$ in Camera 2 frame.

'O' origin of World Coordinate System

(a)

P_1 (12,0) in Camera 1
P_2 (1,0) in Camera 2



We have $(x, y) = (0, 0)$ origin of Camera 1 in Word Coordinate given by $(-20, 0, 0) \rightarrow O_1$

$(u, v) = (0, 0)$ origin of Camera 2 in Word Coordinate System $(20, 0, 0) \rightarrow O_2$

$\therefore$ Triangle $AP'B'$ similar to ACB

$$\therefore \frac{x - (-20)}{12} = \frac{z}{1}$$

$$x + 20 = 12z \quad \ldots\ldots (i)$$

Triangle $QB''P_2$ similar to QBC

$$\therefore \frac{(x-20)}{1} = \frac{z}{1}$$

$$\therefore x = 20 + z \quad \ldots\ldots (ii)$$

Solving (i) and (ii) we get:

$$x = \frac{260}{11} \quad ; \quad z = \frac{40}{11}$$

for Calculating 'y' we Can Similarly use Similar triangle property for triangles along y-axis: using Point P_2'

$$\frac{y-0}{12} = \frac{2-0}{1}$$

$$y = \frac{480}{11}$$

Hence, 3-D location of this point -

$$P(x,y,z) = \left(\frac{260}{11}, \frac{480}{11}, \frac{40}{11}\right)$$

(b) For any point on the line $x+z=0$ we have point represented by $(\alpha, 0, -\alpha)$ which lies on it. Using previous setup and Similar triangles we have!

$$\frac{\alpha-(-20)}{x} = -\alpha \longrightarrow \text{Camera 1}$$

$$\frac{\alpha-20}{u} = -\alpha \longrightarrow \text{Camera 2}$$

$$\alpha = \left(\frac{20}{u+1}\right) \quad \text{and} \quad x = \frac{-(\alpha+20)}{\alpha}$$

or $u = \frac{-(\alpha-20)}{\alpha}$

Q.1.

(b) Continuation!.

Since disparity 'd'

$d = x - u$ (Using x & u in terms of α from previous.)

$= -\frac{(\alpha+20)}{\alpha} + \frac{\alpha-20}{\alpha}$

$= -\frac{40}{\alpha}$

$\therefore \quad = \frac{-40}{20}(u+1)$ put $\alpha = \frac{20}{u+1}$

$$\boxed{d = -2(u+1)}$$

disparity.