SIFT 対応点検出

三田村彰大

2026 年 6 月 12 日

[1]:
```
import cv2
import numpy as np
from pathlib import Path
import matplotlib.pyplot as plt

import shutil

def clear_directory(dirpath):
    dirpath = Path(dirpath)
    for p in dirpath.iterdir():
        if p.is_file() or p.is_symlink():
            p.unlink()
        elif p.is_dir():
            shutil.rmtree(p)
```

0.0.1 画像の読み込み・前処理

$$\text{画像} = I(x, y)$$

今回は 4038×3024 pix 画像となる。また、$0 \sim 255$ を正規化して $0 \sim 1$ にする。（すなわち

$$x, y \in M([\mathbb{0}, \mathbb{1}])^{4038 \times 3024}$$

）

[2]:
```
def read_image(path,max_size):
    # pix*pix の 0~1 の numpy 配列として返す。
    img_read=cv2.imread(str(path),cv2.IMREAD_GRAYSCALE)
    cv2.imwrite(Path(path).stem+"_gray"+Path(path).suffix,img_read)
    img=np.array(img_read, dtype=np.float32) / 255.0
    print(img.shape)
    max_size=max(max_size,max(img.shape))
    return img,max_size


max_size=0
img1,max_size = read_image("./img1.png",max_size)
img2,max_size = read_image("./img2.png",max_size)
```

```
(4032, 3024)
(4032, 3024)
```

0.0.2 DoG を用いた特徴点検出

大きさ $k^i \times \sigma_0$ のガウシアンフィルタ $G(k\sigma_0), G(k^2\sigma_0), G(k^3\sigma_0) \ldots$ に対しそれらによるフィルタリング $G(k\sigma_0) * I, G(k^2\sigma_0) * I, G(k^3\sigma_0) * I \ldots$ を考える。この時、あるスケール $k^n\sigma_0$ において (x, y) に特徴点があるならば、$G(k^n\sigma_0) * I - G(k^{n\pm1}\sigma_0) * I$ の極値として現れるはずである。よって畳み込みの線形性から、

$$\mathrm{DoG}(k^n\sigma_0) = G(k^{n+1}\sigma_0) - G(k^n\sigma_0)$$

というフィルタでフィルタリングを行えばよく、$\mathrm{DoG}(\sigma) * I$ に現れる極地をスケール σ における特徴点とする。(すなわち特徴量は (x, y, σ) としてその位置とスケールが検出される。)

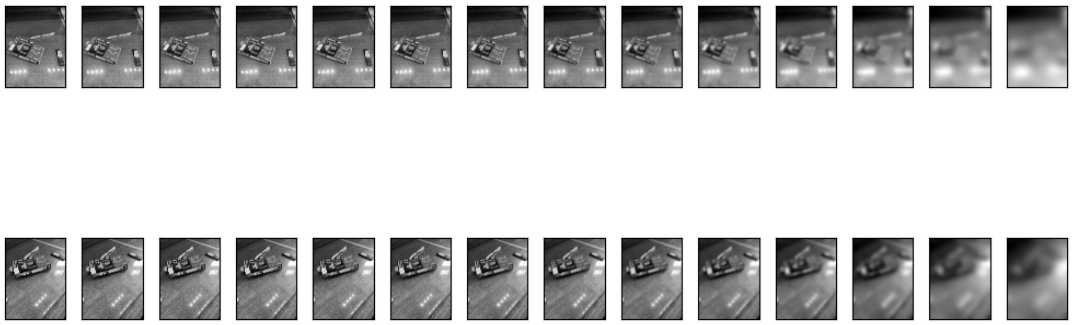
[3]:
```
from scipy.ndimage import gaussian_filter

def generate_sigmas(sigma0,k,max_size,max_sigma=1e10):
    n=0
    sigmas=[]
    if (max_sigma < max_size):
        max_size=max_sigma
    while True:
        sigma = (k**n)*sigma0
        if (sigma > max_size):
            break

        sigmas.append((k**n)*sigma0)
        n=n+1
    return sigmas

sigmas = generate_sigmas(sigma0=1,k=1.5,max_size=max_size,max_sigma=250)
print(sigmas)

def Gaussian_filters(img,sigmas,outdir):
    # sigmas = [sigma1,sigma2,...]に対し G(sigma1)*I,G(sigma2)*I,...を出力
    outdir = Path(outdir)
    outdir.mkdir(parents=True, exist_ok=True)
    clear_directory(outdir)
    gaussianed = []

    for i,sigma in enumerate(sigmas):
        blurred = gaussian_filter(img,sigma)
        gaussianed.append(blurred)

        outpath=outdir / f"gaussian_{i}_sigma{sigma:.3f}.png"
        #print(outpath)
        blurred=(blurred*255).astype(np.uint8)
        cv2.imwrite(outpath,blurred)

    return gaussianed

def Dog_filters(images,outdir):

    outdir = Path(outdir)
    doged_images = []

    for i in range(len(images)-1):
        doged=images[i+1]-images[i]
        doged_images.append(doged)

        outpath=outdir / f"dogged_{i}.png"
        #マイナスの値も取りうるので、正規化
        doged = (doged - doged.min())/(doged.max() - doged.min())

        doged=(doged*255).astype(np.uint8)
        cv2.imwrite(outpath,doged)

    return doged_images

def images_print(images):
```

```
    fig=plt.figure(figsize=(10,3))
    for i in range(len(images)):
        img_out = images[i]
        ax = fig.add_subplot(1, len(images), i+1, xticks=[], yticks=[])
        ax.imshow(img_out,'gray')
    fig.tight_layout()
    plt.show()
```

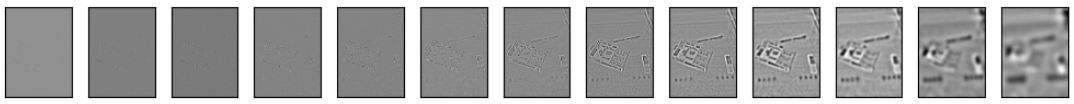
```
[1.0, 1.5, 2.25, 3.375, 5.0625, 7.59375, 11.390625, 17.0859375, 25.62890625,
38.443359375, 57.6650390625, 86.49755859375, 129.746337890625,
194.6195068359375]
```

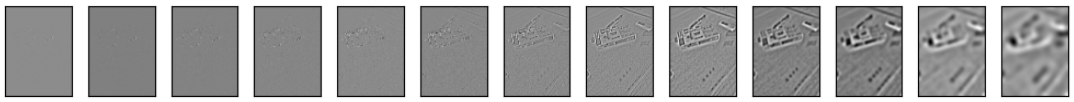
[4]:
```
img1_filtered=Gaussian_filters(img1,sigmas,"img1")
img2_filtered=Gaussian_filters(img2,sigmas,"img2")
print("After Gaussian filter")
images_print(img1_filtered)
images_print(img2_filtered)

img1_doged=Dog_filters(img1_filtered,"img1")
img2_doged=Dog_filters(img2_filtered,"img2")

print("After Dog filter")
images_print(img1_doged)
images_print(img2_doged)
```

```
After Gaussian filter
```




```
After Dog filter
```





0.0.3 特徴量検出

DoG フィルター後、大きさ $H \times W$ の画像が S 枚あるとすると（ただし σ の大きさ順に並べる）$S \times H \times W$ のテンソルに対し極大・極小を与える点が (σ, x, y) 上の特徴点。（ただし $(\sigma, x, y) \leftrightarrow (S, H, W)$ 、授業スライド参照。）今回は $3 \times 3 \times 3$ の max/min filter で探査する。（授業スライド参照。）

また、極値を与えたとしても関値を超えないくらいのものは無視する。(0.03 くらいがいいらしい (?)、しょうもない点を検出しないため。) あと画像端も無視する。

その上で、極値の値が上位のものを採用する。上位何個の特徴点を採用すればいいかは、結果を見ながら考える。)

[5]:
```python
from scipy.ndimage import maximum_filter, minimum_filter

def find_keypoints(imges_doged,threshhold,sigmas,max_points):

    image_stack = np.stack(imges_doged,axis=0)

    #max/min_filter は最大値、最小値で置き換える。
    max_filtered_stack = maximum_filter(image_stack, size=(3, 3, 3))
    min_filtered_stack = minimum_filter(image_stack, size=(3, 3, 3))
    is_max = (max_filtered_stack==image_stack)
    is_min = (min_filtered_stack==image_stack)

    extreme_points = (is_max | is_min) & (np.abs(image_stack) > threshhold)
    extreme_points[0, :, :] = False
    extreme_points[-1, :, :] = False
    extreme_points[:, 0, :] = False
    extreme_points[:, -1, :] = False
    extreme_points[:, :, 0] = False
    extreme_points[:, :, -1] = False

    #局地を与える x,y,sigma の列（画像は転置してるので注意！）
    sigma_i,y,x = np.where(extreme_points==True)
    values = np.abs(image_stack[sigma_i, y, x])
    order = np.argsort(values)[::-1]
    order = order[:max_points]
    sigma_i = sigma_i[order]
    y = y[order]
    x = x[order]

    keypoints_coord=[]
    for sigma_i,y,x in zip(sigma_i,y,x):
        sigma = sigmas[sigma_i]
        keypoints_coord.append((x,y,sigma))

    return keypoints_coord

def show_keypoints(img,keypoints_coord):
    plt.figure(figsize=(8, 8))
    plt.imshow(img, cmap="gray")
    xs = [kp[0] for kp in keypoints_coord]
    ys = [kp[1] for kp in keypoints_coord]

    plt.scatter(xs,ys,2.0,facecolors="none",edgecolors="r",linewidths=1.0)
    plt.axis("off")
    plt.tight_layout()

    #if out_path is not None:
        #plt.savefig(out_path, dpi=200, bbox_inches="tight", pad_inches=0)

    plt.show()
```

[6]:
```
max_points=5000

img1_keypoints = find_keypoints(img1_doged,threshhold=0.03,sigmas=sigmas,max_points=max_points)
img2_keypoints = find_keypoints(img2_doged,threshhold=0.03,sigmas=sigmas,max_points=max_points)

show_keypoints(img1,img1_keypoints)
show_keypoints(img2,img2_keypoints)
```

0.0.4　SIFT 記述子

今画像座標を X, Y で表すことにする。この時まず準備として、各スケール σ での平滑化画像 $L^\sigma(X,Y)$ に対し、

$$m(X,Y) = ||\nabla L(X,Y)||_2 \quad , \quad \theta(X,Y) = \arg\{\nabla L(X,Y)\}$$

を計算しておく。

その上で、座標 X_0, Y_0 で検出された特徴点を記述することを考える。この時まず、特徴点のスケール σ（今回は $\mathrm{DoG}(k^n\sigma_0) = G(k^{n+1}\sigma_0) - G(k^n\sigma_0)$ で検出された特徴点のスケールをそのまま $\sigma = k^n\sigma_0$ としている。）による平滑化画像 $L^\sigma(X,Y)$ を使い、(X_0, Y_0) 近傍での代表的な方向 θ_0 を決定する必要がある。

Lowe の方法によればこれは 2π を 36 分割したヒストグラムに対し、$\theta(X,Y)$　を $G(X - X_0, Y - Y_0, 1.5\sigma)$ 及び $m(X,Y)$ によって重み付けした上で投票することによって決定される。（また、論文には記載がなかったが、ふたつの投票 bin に対して分散投票を行う。（各投票 bin への近さに応じて重み付けしてそれぞれ投票を行う。））このようにして決定された方向を今 θ_0 とすることにする。ただし、

“The highest peak in the histogram is detected, and then any other local peak that is within 80% of the highest peak is used to also create a keypoint with that orientation.”

とあることから、一つの特徴点に対し複数の方向 θ_0 が与えられることも起こりうる。

代表的方向が決定できたら次は特徴点近傍で輝度勾配の投票を行う。Lowe の論文で

“In order to achieve orientation invariance, the coordinates of the descriptor and the gradient orientations are rotated relative to the keypoint orientation”

とあるように、そのためには特徴点近傍で θ_0 が主軸となるような座標系をとり、さらに $\theta \rightsquigarrow \theta - \theta_0$ のような変換を行う必要がある。また最終的には正規化された座標系（授業のような正規化を考え、$-1.5, -0.5, 0.5, 1.5$ が bin 中心となるようにする）で考えたいので、座標のスケールも変換する必要がある。よって、

$$\begin{pmatrix} x \\ y \end{pmatrix} = \frac{1}{s} R(-\theta_0) \begin{pmatrix} X - X_0 \\ Y - Y_0 \end{pmatrix}$$

（すなわち $(x,y) = f(X,Y)$ ）のように変換し、輝度勾配についても

$$\begin{aligned} m\,(x,y) &= m\,(X,Y) && (\text{すなわち } = m\,(f^{-1}(x,y))\,) \\ \theta(x,y) &= \theta(X,Y) - \theta_0 && (\text{すなわち } = \theta\,(f^{-1}(x,y)) - \theta_0\,) \end{aligned}$$

のように変換しなければならない。（ここら辺は論文の行間を読んだり AI と相談したりしながら考えた。）ただし s については、例えば授業スライドのような一つの空間方向の bin あたりの pix 数が 4 となるような場

合を考えると $s = 4$ となる。（ただし、本当は空間方向の bin あたりの pix 数も特徴量のスケールに応じて変化させるべきであり、$s = 4\sigma$ などとするべきであると思われる。）

その上で、このようにして準備した (x, y) さえ用意できれば、あとは授業で示された投票式に従って

$$v(i,j,k) = \sum_{(x,y)\in\mathbb{R}} G(x,y,\sigma_w)m(x,y)w_p(x-x_i)w_p(y-y_j)w_a(\theta(x,y)-\theta_k)$$

と投票すれば良い。ただしこのとき、$G(x, y, \sigma_w)$ の σ_w の取り方については Lowe の文献 descriptor window の半分としているので、今回の場合 $\sigma_w = 2$ と取ることになる。

また、

“Therefore, we reduce the influence of large gradient magnitudes by thresholding the values in the unit feature vector to each be no larger than 0.2, and then renormalizing to unit length.”

とあるように、照明の非線形変化 (?) の影響をなくすためには特徴量ベクトルをクリッピングするといいらしい。

[7]:
```python
#輝度勾配の算出
def compute_gradients(L):
    # Gaussian filtered image L^\sigma から勾配強度 m と勾配方向 theta を計算する。
    L = L.astype(np.float32)

    dx = np.zeros_like(L, dtype=np.float32)
    dy = np.zeros_like(L, dtype=np.float32)

    # 勾配は中心差分で取る。
    dx[:, 1:-1] = (L[:, 2:] - L[:, :-2])/2.0
    dy[1:-1, :] = (L[2:, :] - L[:-2, :])/2.0

    magnitude = np.sqrt(dx * dx + dy * dy)
    orientation = (np.arctan2(dy, dx) + 2.0 * np.pi) % (2.0 * np.pi)

    return magnitude, orientation

#代表方向の決定
def find_main_orientation(X0,Y0,sigma,magnitude,orientation):
    # X0,Y0 周りの点を使って、ガウシアン重み+ m(X,Y) 重みを使いながら投票する。
    num_bins=36
    h, w = magnitude.shape
    vote_sigma = 1.5 * sigma
    radius = int(round(3.0 * vote_sigma))
    x_min = max(1, X0 - radius)
    x_max = min(w - 1, X0 + radius)
    y_min = max(1, Y0 - radius)
    y_max = min(h - 1, Y0 + radius)

    hist = np.zeros(num_bins,dtype=np.float32)
    for Y in range(y_min, y_max):
        for X in range(x_min,x_max):
            dX=X-X0
            dY=Y-Y0
            r2=dX*dX+dY*dY
            if (r2>radius*radius):
                continue

            gaussian_weight = np.exp(-r2 /(2.0*vote_sigma*vote_sigma))
            vote_weight = gaussian_weight * magnitude[Y, X]
            bin_float= (orientation[Y, X] * num_bins/(2.0 * np.pi)) % num_bins
            bin_to_vote = int(np.floor(bin_float))
            db = bin_float - bin_to_vote
            hist[bin_to_vote] += vote_weight * (1.0 - db)
            hist[(bin_to_vote + 1) % num_bins] += vote_weight * db
```

```
    main_orientations=[]
    max_val = hist.max()

    for bin in range(num_bins):
        left_hist_val = hist[(bin - 1) % num_bins]
        center_hist_val = hist[bin]
        right_hist_val = hist[(bin + 1) % num_bins]

        if ((center_hist_val >= left_hist_val) and (center_hist_val >= right_hist_val) and center_hist_val >= 0.8 *␣
↪max_val):
            theta = ((2.0 * np.pi) * bin/ num_bins) % (2.0 * np.pi)
            main_orientations.append(theta)

    return main_orientations

#三角重み
def triangular_weight(z):
    return max(0.0, 1.0 - abs(z))

#角度用の三角重み（授業スライドのやつは、多肢性を考慮したり角度の単位を正規化してるだけで三角重みとやってることは同じ。）
def circular_bin_weight(bin_pos, bin_center, num_bins):
    d = abs(bin_pos - bin_center)
    d = min(d, num_bins - d)
    return max(0.0, 1.0 - d)

#128次元 descriptorの作成
def compute_descriptor_one_keypoint(X0,Y0,sigma,theta0,magnitude, orientation):

    spatial_bins=4
    orientation_bins=8
    bin_scale=4.0
    sigma_w=2.0

    #X0, Y0:画像座標上の特徴点位置
    #sigma:DoGで得た特徴点スケール
    #theta0:代表方向
    #magnitude, orientation: 対応スケールの Gaussian画像から計算した勾配強度・勾配方向
    #bin_scale:1つの空間 binが画像上で何 sigma分の幅を持つかで、s = bin_scale * sigma となる。
    #sigma_w:descriptor座標系でのガウシアン重みの標準偏差。4×4 bin全体の幅が 4なので、半分の 2.0を使う。

    h, w = magnitude.shape
    X0 = float(X0)
    Y0 = float(Y0)
    s = bin_scale * sigma

    #ヒストグラム
    centers = np.arange(spatial_bins, dtype=np.float32) - (spatial_bins - 1) / 2.0
    hist = np.zeros((spatial_bins, spatial_bins, orientation_bins), dtype=np.float32)
    #投票での探索範囲
    descriptor_radius_local = (spatial_bins/2.0) + 0.5

    #座標変換まわり
    cos_theta = np.cos(theta0)
    sin_theta = np.sin(theta0)
    #画像座標 (X,Y)上での探索範囲
    radius = int(np.ceil(np.sqrt(2.0) * descriptor_radius_local * s))
    X_min = max(1, int(round(X0)) - radius)
    X_max = min(w - 2, int(round(X0)) + radius)
    Y_min = max(1, int(round(Y0)) - radius)
    Y_max = min(h - 2, int(round(Y0)) + radius)

    for Y in range(Y_min, Y_max + 1):
        for X in range(X_min, X_max + 1):
            dX = X - X0
            dY = Y - Y0
            #座標変換
            x = ( cos_theta * dX + sin_theta * dY) / s
            y = (-sin_theta * dX + cos_theta * dY) / s
            phi = (orientation[Y, X] - theta0) % (2.0 * np.pi)
            # 角度を bin単位に変換
            angle_bin_pos = phi * orientation_bins / (2.0 * np.pi)

            #投票
            if (abs(x) >= descriptor_radius_local or abs(y) >= descriptor_radius_local):
                continue
            gaussian_weight = np.exp(-(x * x + y * y) / (2.0 * sigma_w * sigma_w))
            for j, y_center in enumerate(centers):
                wy = triangular_weight(y - y_center)
                if wy <= 0:
```

```
                    continue

                for i, x_center in enumerate(centers):
                    wx = triangular_weight(x - x_center)
                    if wx <= 0:
                        continue

                    for k in range(orientation_bins):
                            wa = circular_bin_weight(angle_bin_pos, k, orientation_bins)
                            if wa <= 0:
                                continue
                            hist[j, i, k] += gaussian_weight * magnitude[Y, X] * wx * wy * wa

    descriptor = hist.reshape(-1)
    norm = np.linalg.norm(descriptor)
    descriptor = descriptor / norm

    descriptor = np.clip(descriptor, 0, 0.2)
    norm = np.linalg.norm(descriptor)
    if norm < 1e-12:
        return None
    descriptor = descriptor / norm

    return descriptor.astype(np.float32)

def precompute_gradients(gaussian_images):
    magnitudes = []
    orientations = []
    for L in gaussian_images:
        mag, ori = compute_gradients(L)
        magnitudes.append(mag)
        orientations.append(ori)
    return magnitudes, orientations

def compute_descriptors(keypoints_coord,gaussian_images,sigmas,max_keypoints):

    magnitudes, orientations = precompute_gradients(gaussian_images)
    valid_keypoints = []
    descriptors = []
    keypoints_coord = keypoints_coord[:max_keypoints]
    sigmas_np = np.array(sigmas, dtype=np.float32)

    for _, (X0, Y0, sigma) in enumerate(keypoints_coord):
        # sigma に一番近い Gaussian 画像を使う
        scale_idx = int(np.argmin(np.abs(sigmas_np - sigma)))

        magnitude = magnitudes[scale_idx]
        orientation = orientations[scale_idx]

        theta0_list = find_main_orientation(X0,Y0,sigma,magnitude,orientation)

        for theta0 in theta0_list:
            descriptor = compute_descriptor_one_keypoint(X0,Y0,sigma,theta0,magnitude,orientation)
            valid_keypoints.append((X0, Y0, sigma, theta0))
            descriptors.append(descriptor)

    descriptors = np.stack(descriptors, axis=0)

    return valid_keypoints, descriptors
```

[8]:
```
img1_valid_keypoints, img1_descriptors = compute_descriptors(img1_keypoints,img1_filtered,sigmas,max_keypoints=1000)

img2_valid_keypoints, img2_descriptors = compute_descriptors(img2_keypoints,img2_filtered,sigmas,max_keypoints=1000,)

print("img1 valid keypoints:", len(img1_valid_keypoints))
print("img1 descriptors:", img1_descriptors.shape)

print("img2 valid keypoints:", len(img2_valid_keypoints))
print("img2 descriptors:", img2_descriptors.shape)
```

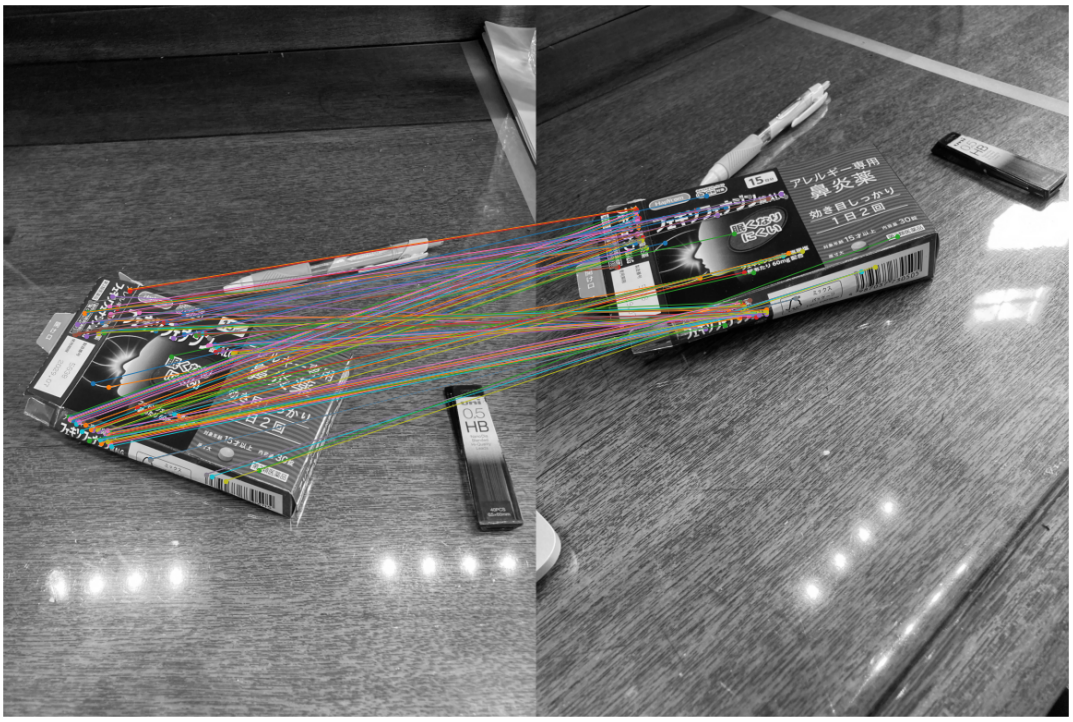
```
img1 valid keypoints: 1461
img1 descriptors: (1461, 128)
img2 valid keypoints: 1463
img2 descriptors: (1463, 128)
```

0.0.5 最近傍探査 +Ratio Test

二つの画像について 128 次元 descriptor d_i^1, d_j^2 得られているとき、$||d_i^1 - d_j^2||$ が最小となるような descriptor のペアをマッチングする。またその際、第二近傍も探索し、ratio test をパスしたものをマッチングペアとして採用する。（すなわち、$\mathrm{dist}(i,j) = ||d_i^1 - d_j^2||$ を最小とする (i_1,j_1) と次点で最小値を与える (i_2, j_2) を探し、$\mathrm{dist}(i_1,j_1)/\mathrm{dist}(i_2,j_2) < r$ となったものをペアとして採用することにする。）こうすることで、辺なマッチングを除外できることが期待される。

[9]:
```
def match_descriptors(descriptor1, descriptor2, ratio):
    matches = []
    #距離行列 (i,j)
    distances = np.linalg.norm(descriptor1[:, None, :] - descriptor2[None, :, :], axis=2)

    for i in range(distances.shape[0]):
        order = np.argsort(distances[i])
        j1 = order[0]
        j2 = order[1]

        d1 = distances[i, j1] #第一近傍
        d2 = distances[i, j2] #第二近傍

        if d1 < ratio * d2:
            matches.append((i, j1, d1, d2))

    return matches

def show_matches(img1, keypoints1, img2, keypoints2, matches, max_matches):
    H1, W1 = img1.shape
    H2, W2 = img2.shape

    H = max(H1, H2)
    W = W1 + W2

    canvas = np.ones((H, W), dtype=np.float32)
    canvas[:H1, :W1] = img1
    canvas[:H2, W1:W1 + W2] = img2

    plt.figure(figsize=(14, 8))
    plt.imshow(canvas, cmap="gray")

    matches_sorted = sorted(matches, key=lambda t: t[2])
    matches_draw = matches_sorted[:max_matches]

    for i, j, d1, d2 in matches_draw:
        X1, Y1, sigma1, theta1 = keypoints1[i]
        X2, Y2, sigma2, theta2 = keypoints2[j]

        plt.plot([X1, X2 + W1], [Y1, Y2], linewidth=0.7)
        plt.scatter([X1, X2 + W1], [Y1, Y2], s=8)

    plt.axis("off")
    plt.tight_layout()
    plt.show()
```

[10]:
```
matches = match_descriptors(img1_descriptors,img2_descriptors,ratio=0.7)
show_matches(img1,img1_valid_keypoints,img2,img2_valid_keypoints,matches,max_matches=1000)
```



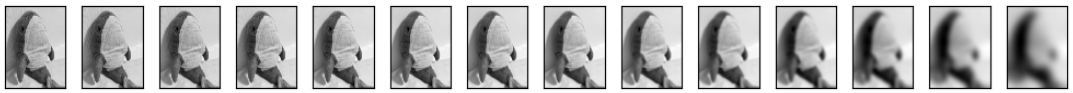
0.0.6 他の画像でも試してみる

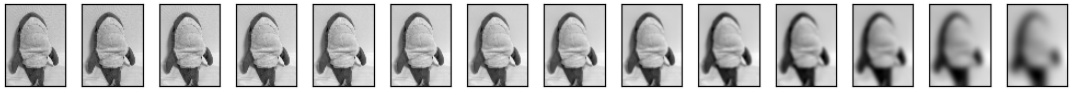
[11]:
```
max_size=0
img3,max_size = read_image("./img3.png",max_size)
img4,max_size = read_image("./img4.png",max_size)
```

```
(4032, 3024)
(4032, 3024)
```

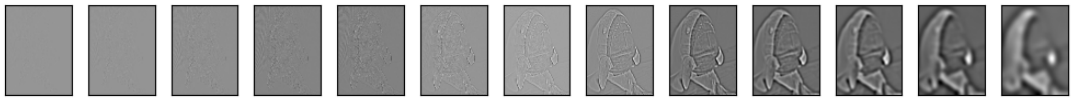
[12]:
```
sigmas = generate_sigmas(sigma0=1,k=1.5,max_size=max_size,max_sigma=250)
print(sigmas)

img3_filtered=Gaussian_filters(img3,sigmas,"img3")
img4_filtered=Gaussian_filters(img4,sigmas,"img4")
print("After Gaussian filter")
images_print(img3_filtered)
images_print(img4_filtered)

img3_doged=Dog_filters(img3_filtered,"img3")
img4_doged=Dog_filters(img4_filtered,"img4")

print("After Dog filter")
images_print(img3_doged)
images_print(img4_doged)
```

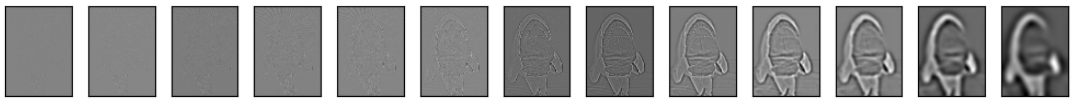
```
[1.0, 1.5, 2.25, 3.375, 5.0625, 7.59375, 11.390625, 17.0859375, 25.62890625,
38.443359375, 57.6650390625, 86.49755859375, 129.746337890625,
194.6195068359375]
After Gaussian filter
```



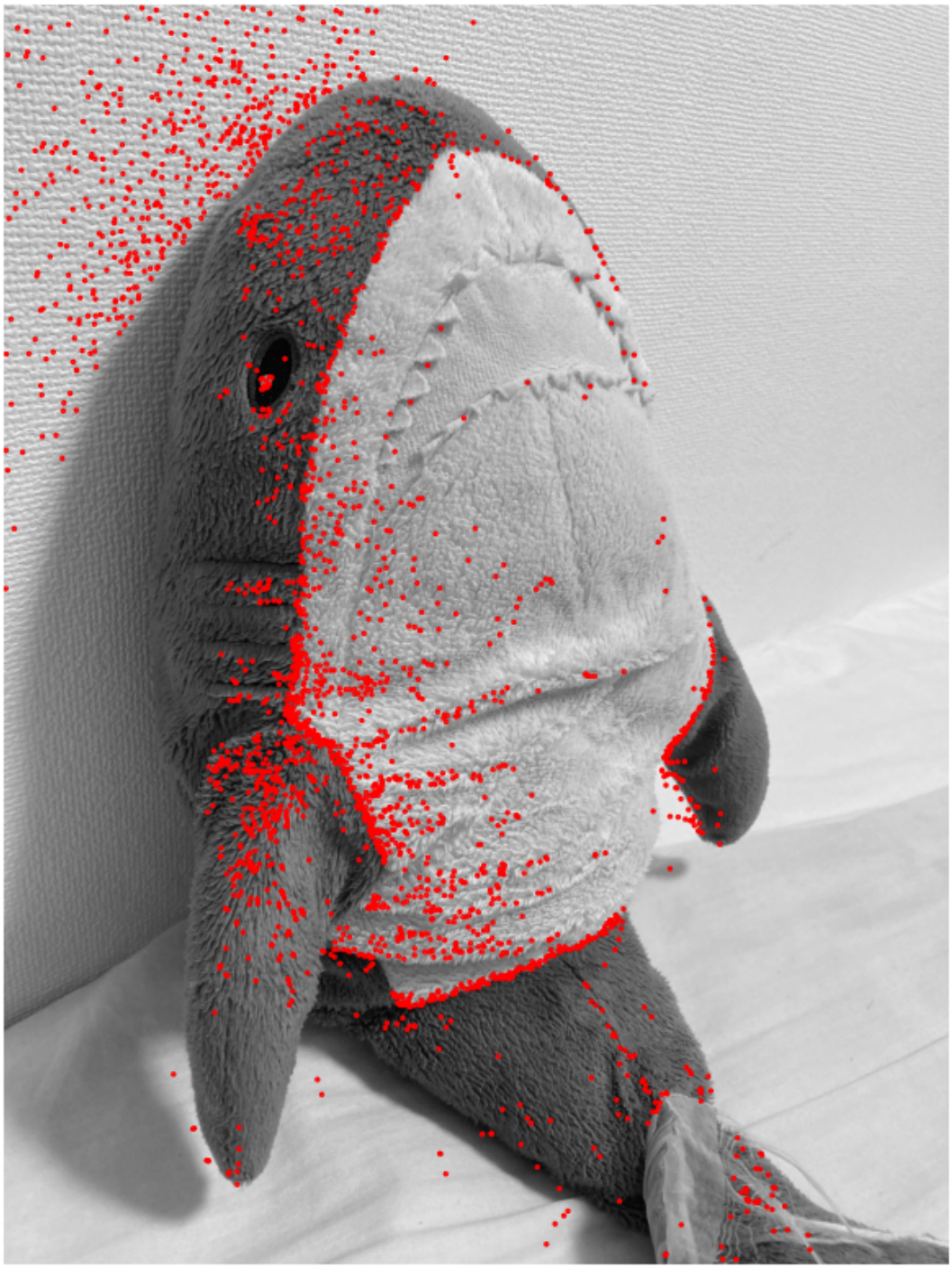


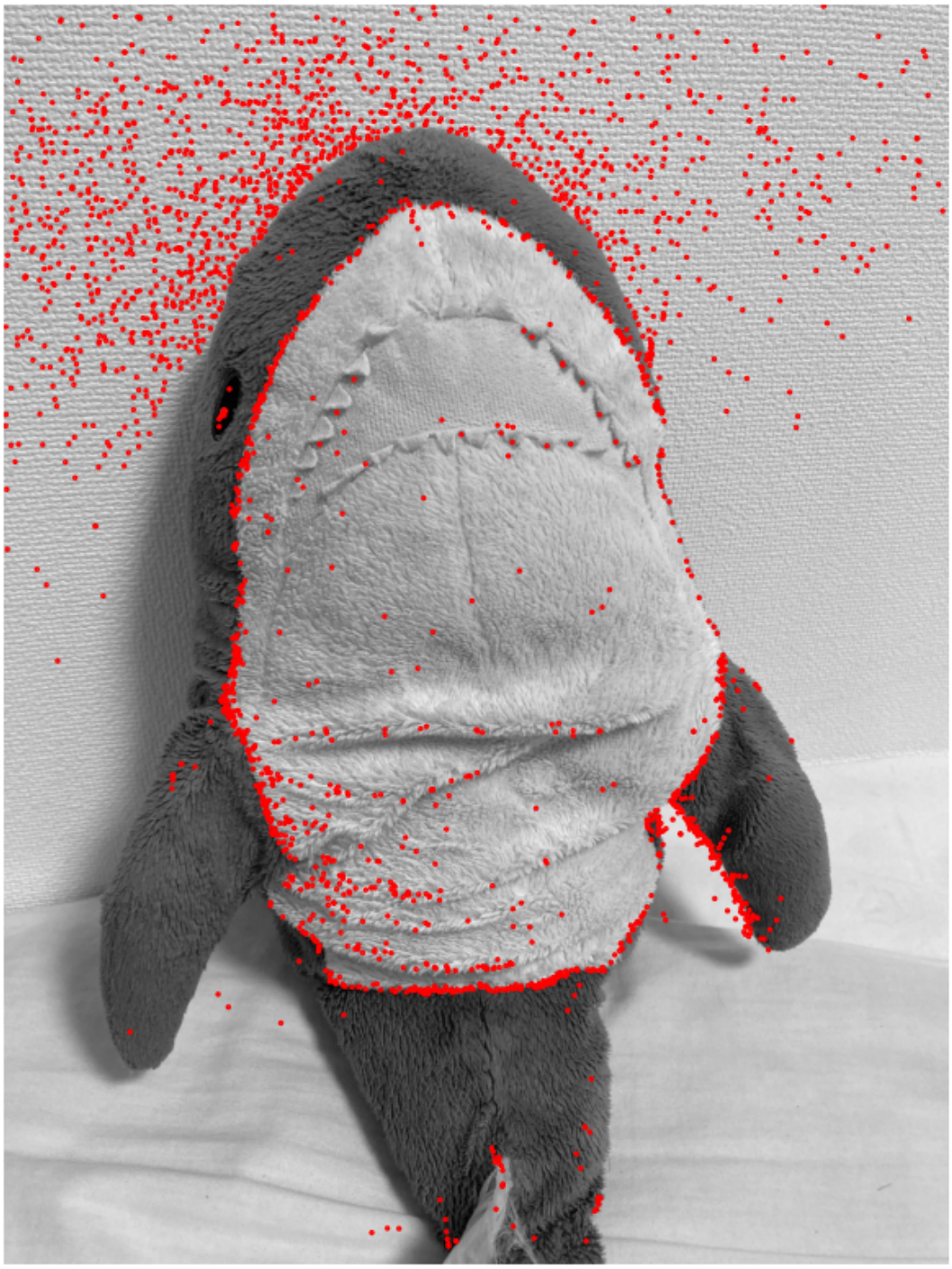
After Dog filter





[13]:
```
max_points=2500

img3_keypoints = find_keypoints(img3_doged,threshhold=0.03,sigmas=sigmas,max_points=max_points)
img4_keypoints = find_keypoints(img4_doged,threshhold=0.03,sigmas=sigmas,max_points=max_points)

show_keypoints(img3,img3_keypoints)
show_keypoints(img4,img4_keypoints)
```



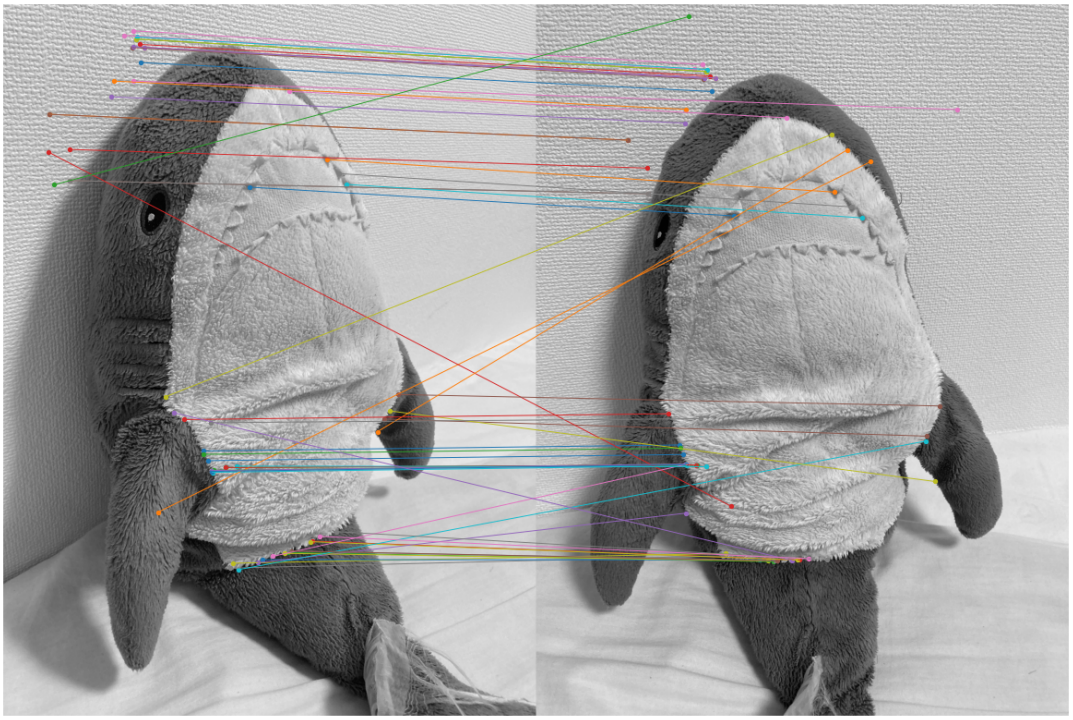


[14]:
```
img3_valid_keypoints, img3_descriptors = compute_descriptors(img3_keypoints,img3_filtered,sigmas,max_keypoints=1000)

img4_valid_keypoints, img4_descriptors = compute_descriptors(img4_keypoints,img4_filtered,sigmas,max_keypoints=1000)
```

```
print("img3 valid keypoints:", len(img3_valid_keypoints))
print("img3 descriptors:", img3_descriptors.shape)

print("img4 valid keypoints:", len(img4_valid_keypoints))
print("img4 descriptors:", img4_descriptors.shape)
```

```
img3 valid keypoints: 1247
img3 descriptors: (1247, 128)
img4 valid keypoints: 1266
img4 descriptors: (1266, 128)
```

[15]:
```
matches = match_descriptors(img3_descriptors,img4_descriptors,ratio=0.8)
show_matches(img3,img3_valid_keypoints,img4,img4_valid_keypoints,matches,max_matches=1000)
```



[ ]:
```
max_size=0
img5,max_size = read_image("./img5.png",max_size)
img6,max_size = read_image("./img6.png",max_size)
```

```
(4032, 3024)
(4032, 3024)
```

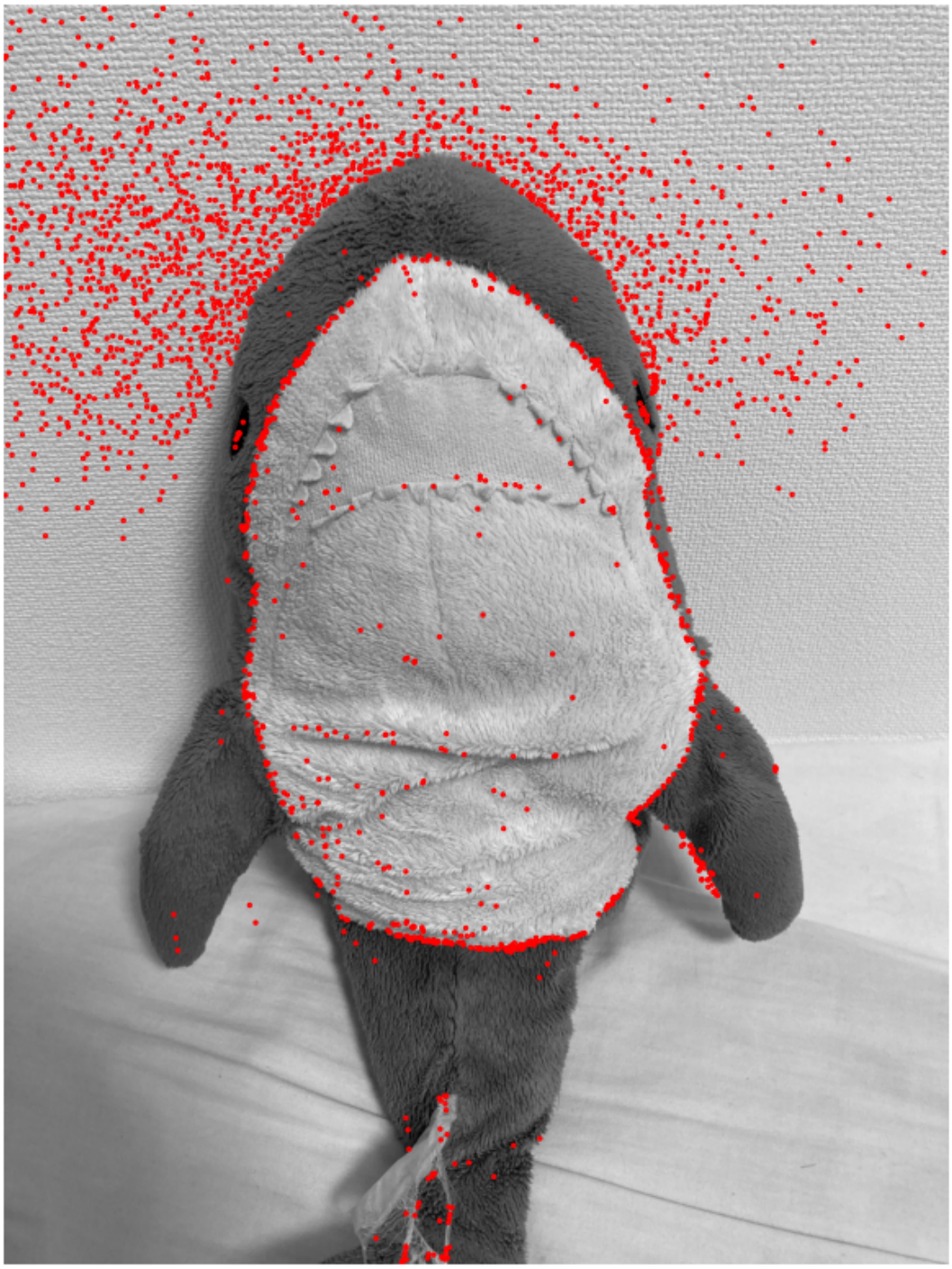
[17]:
```
sigmas = generate_sigmas(sigma0=1,k=1.5,max_size=max_size,max_sigma=250)
print(sigmas)

img5_filtered=Gaussian_filters(img5,sigmas,"img5")
img6_filtered=Gaussian_filters(img6,sigmas,"img6")
print("After Gaussian filter")
images_print(img5_filtered)
images_print(img6_filtered)
```

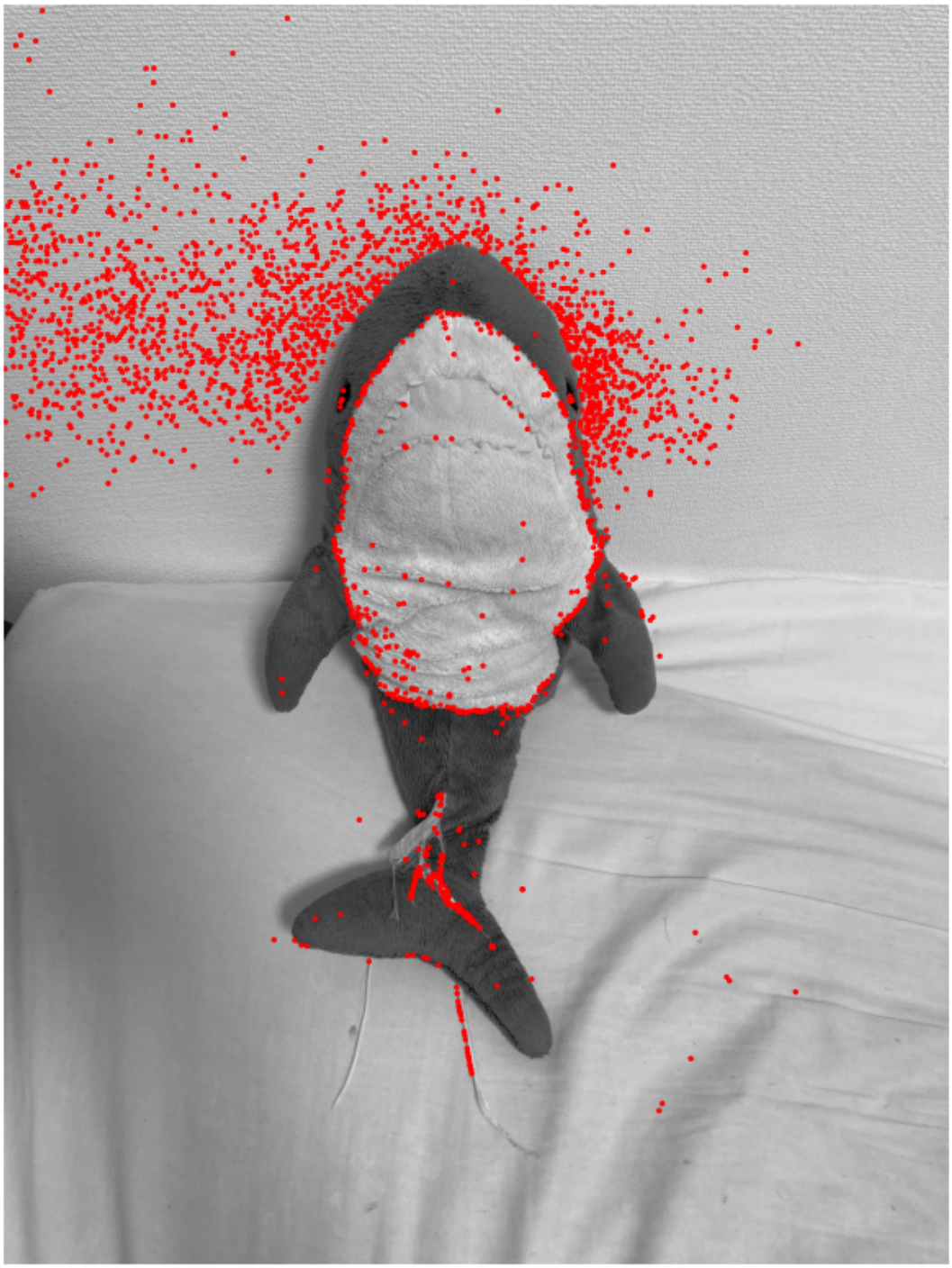
```
img5_doged=Dog_filters(img5_filtered,"img5")
img6_doged=Dog_filters(img6_filtered,"img6")

print("After Dog filter")
images_print(img5_doged)
images_print(img6_doged)
```

```
[1.0, 1.5, 2.25, 3.375, 5.0625, 7.59375, 11.390625, 17.0859375, 25.62890625,
38.443359375, 57.6650390625, 86.49755859375, 129.746337890625,
194.6195068359375]
After Gaussian filter
```

After Dog filter

[18]:
```
max_points=2500

img5_keypoints = find_keypoints(img5_doged,threshhold=0.03,sigmas=sigmas,max_points=max_points)
img6_keypoints = find_keypoints(img6_doged,threshhold=0.03,sigmas=sigmas,max_points=max_points)

show_keypoints(img5,img5_keypoints)
show_keypoints(img6,img6_keypoints)
```





[19]:
```
img5_valid_keypoints, img5_descriptors = compute_descriptors(img5_keypoints,img5_filtered,sigmas,max_keypoints=1000)
img6_valid_keypoints, img6_descriptors = compute_descriptors(img6_keypoints,img6_filtered,sigmas,max_keypoints=1000)
```

```
print("img5 valid keypoints:", len(img5_valid_keypoints))
print("img6 descriptors:", img6_descriptors.shape)

print("img5 valid keypoints:", len(img5_valid_keypoints))
print("img6 descriptors:", img6_descriptors.shape)
```

```
img5 valid keypoints: 1330
img6 descriptors: (1381, 128)
img5 valid keypoints: 1330
img6 descriptors: (1381, 128)
```

[20]:
```
matches = match_descriptors(img5_descriptors,img6_descriptors,ratio=0.8)
show_matches(img5,img5_valid_keypoints,img6,img6_valid_keypoints,matches,max_matches=1000)
```

[ ]:
```
del img1,img1_descriptors,img1_doged,img1_filtered,img1_keypoints,img1_valid_keypoints
del img2,img2_descriptors,img2_doged,img2_filtered,img2_keypoints,img2_valid_keypoints
del img3,img3_descriptors,img3_doged,img3_filtered,img3_keypoints,img3_valid_keypoints
del img4,img4_descriptors,img4_doged,img4_filtered,img4_keypoints,img4_valid_keypoints
del img5,img5_descriptors,img5_doged,img5_filtered,img5_keypoints,img5_valid_keypoints
del img6,img6_descriptors,img6_doged,img6_filtered,img6_keypoints,img6_valid_keypoints
```

[26]:
```
import gc
gc.collect()
```

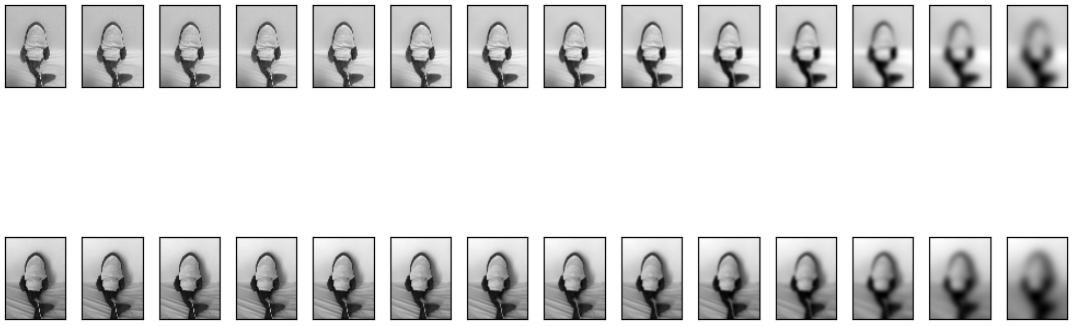
[26]:
```
103260
```

[27]:
```
max_size=0
img7,max_size = read_image("./img7.png",max_size)
img8,max_size = read_image("./img8.png",max_size)
```

```
(4032, 3024)
(4032, 3024)
```

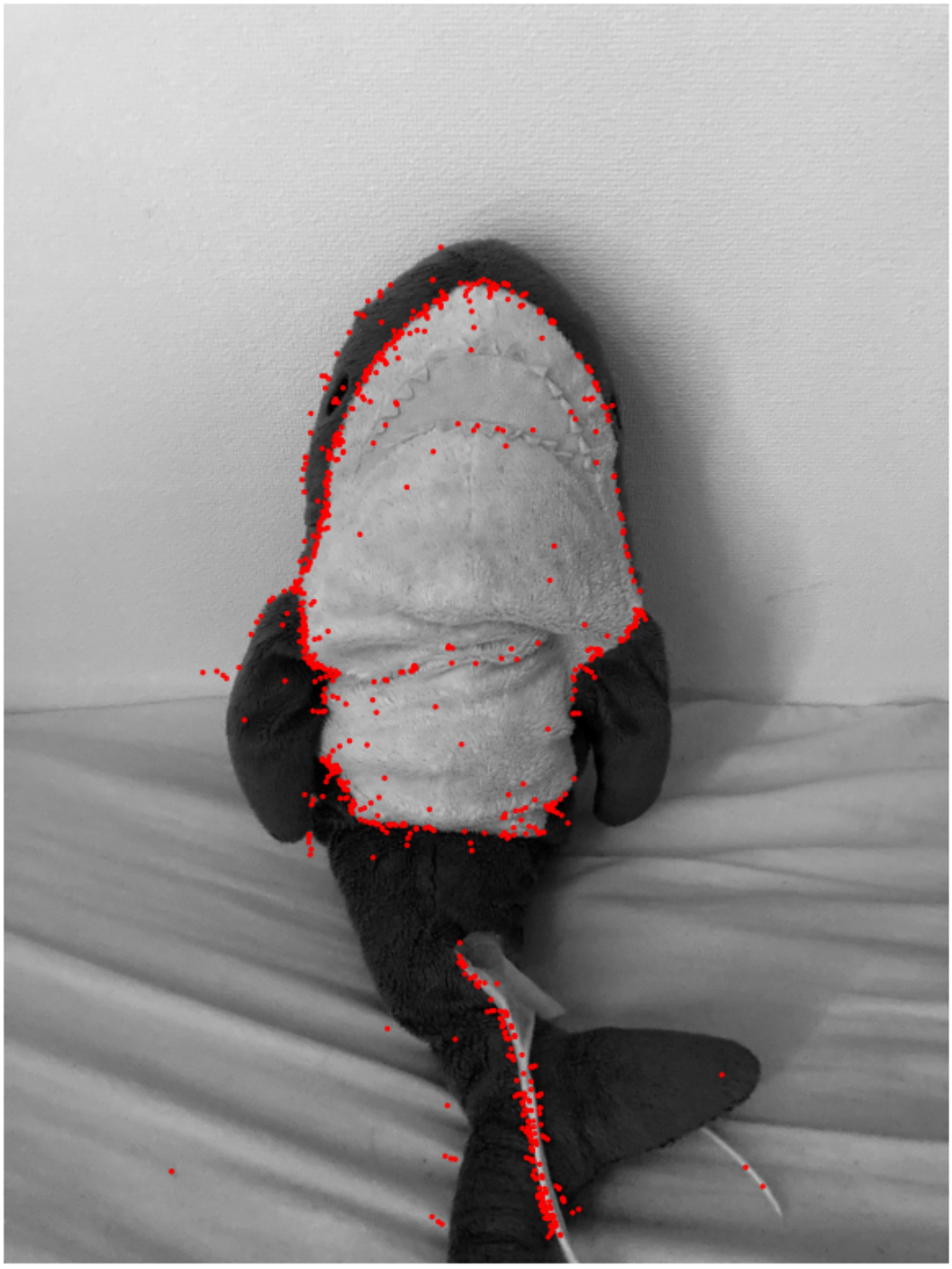
[28]:
```
sigmas = generate_sigmas(sigma0=1,k=1.5,max_size=max_size,max_sigma=250)
print(sigmas)

img7_filtered=Gaussian_filters(img7,sigmas,"img7")
img8_filtered=Gaussian_filters(img8,sigmas,"img8")
print("After Gaussian filter")
images_print(img7_filtered)
images_print(img8_filtered)

img7_doged=Dog_filters(img7_filtered,"img7")
img8_doged=Dog_filters(img8_filtered,"img8")

print("After Dog filter")
images_print(img7_doged)
images_print(img8_doged)
```

```
[1.0, 1.5, 2.25, 3.375, 5.0625, 7.59375, 11.390625, 17.0859375, 25.62890625,
38.443359375, 57.6650390625, 86.49755859375, 129.746337890625,
194.6195068359375]
After Gaussian filter
```



```
After Dog filter
```

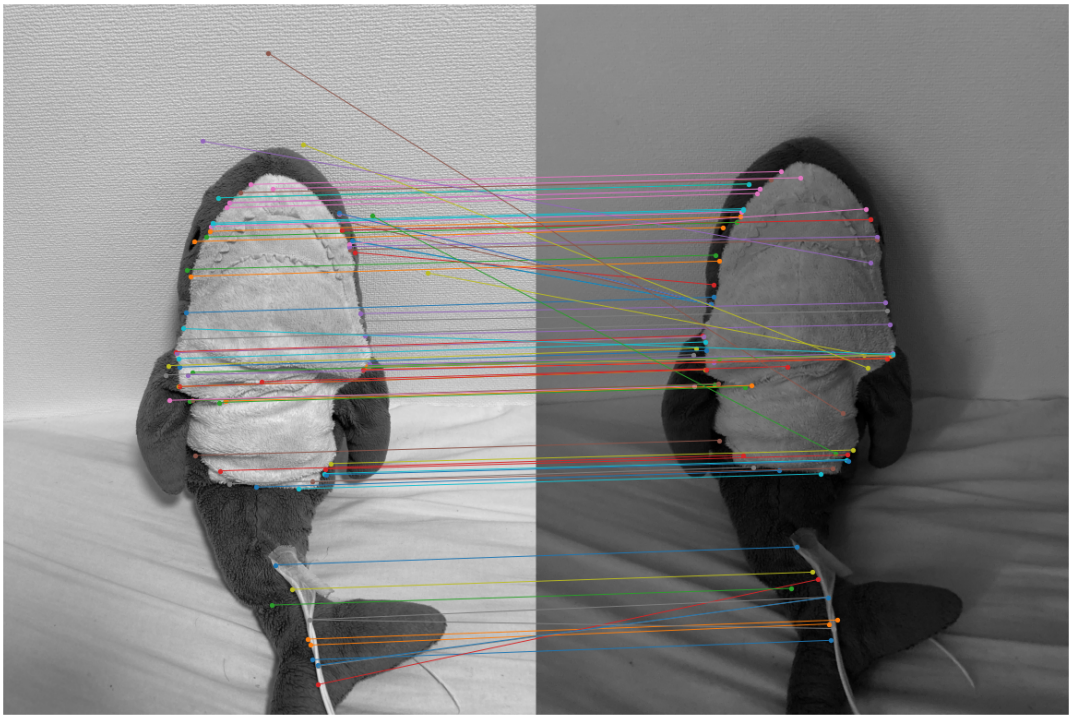
[29]:
```
max_points=2500

img7_keypoints = find_keypoints(img7_doged,threshhold=0.03,sigmas=sigmas,max_points=max_points)
img8_keypoints = find_keypoints(img8_doged,threshhold=0.03,sigmas=sigmas,max_points=max_points)

show_keypoints(img7,img7_keypoints)
show_keypoints(img8,img8_keypoints)
```



[30]:
```
img7_valid_keypoints, img7_descriptors = compute_descriptors(img7_keypoints,img7_filtered,sigmas,max_keypoints=1000)

img8_valid_keypoints, img8_descriptors = compute_descriptors(img8_keypoints,img8_filtered,sigmas,max_keypoints=1000)

print("img7 valid keypoints:", len(img7_valid_keypoints))
print("img8 descriptors:", img8_descriptors.shape)

print("img7 valid keypoints:", len(img7_valid_keypoints))
print("img8 descriptors:", img8_descriptors.shape)
```

```
img7 valid keypoints: 1289
img8 descriptors: (928, 128)
img7 valid keypoints: 1289
img8 descriptors: (928, 128)
```

[31]:
```
matches = match_descriptors(img7_descriptors,img8_descriptors,ratio=0.8)
show_matches(img7,img7_valid_keypoints,img8,img8_valid_keypoints,matches,max_matches=1000)
```



[32]:
```
del img7,img7_descriptors,img7_doged,img7_filtered,img7_keypoints,img7_valid_keypoints
del img8,img8_descriptors,img8_doged,img8_filtered,img8_keypoints,img8_valid_keypoints
gc.collect()
```

[32]:
```
108188
```