pick-spots

March 27, 2019

```
In [1]: import cv2
        from PIL import Image
        import tifffile as tiff
        import matplotlib.pyplot as plt
        from skimage import filters
```

1 Notes

- Install opencv and opencv-contrib. Both versions have to be lower than 3.4.3.
- Range of uint16: [0, 65535]

```
In [2]: import numpy as np
        import bisect

        def imghist(img):
            binrange = [np.min(img), np.max(img)]
            binlength = binrange[1] - binrange[0]
            hist,bins = np.histogram(img.flatten(),binlength, binrange)
            cdf = hist.cumsum()
            cdf_normalized = cdf * hist.max()/ cdf.max()
            plt.plot(cdf_normalized, color = 'b')
            plt.hist(img.flatten(), binlength, binrange, color = 'r')
            plt.xlim(binrange)
            plt.legend(('cdf','histogram'), loc = 'upper left')
            plt.show()

        def imadjust(src, tol=1, vout=(0,255)):
            # src : input one-layer image (numpy array)
            # tol : tolerance, from 0 to 100.
            # vin  : src image bounds
            # vout : dst image bounds
            # return : output img

            assert len(src.shape) == 2 ,'Input image should be 2-dims'

            tol = max(0, min(100, tol))
```

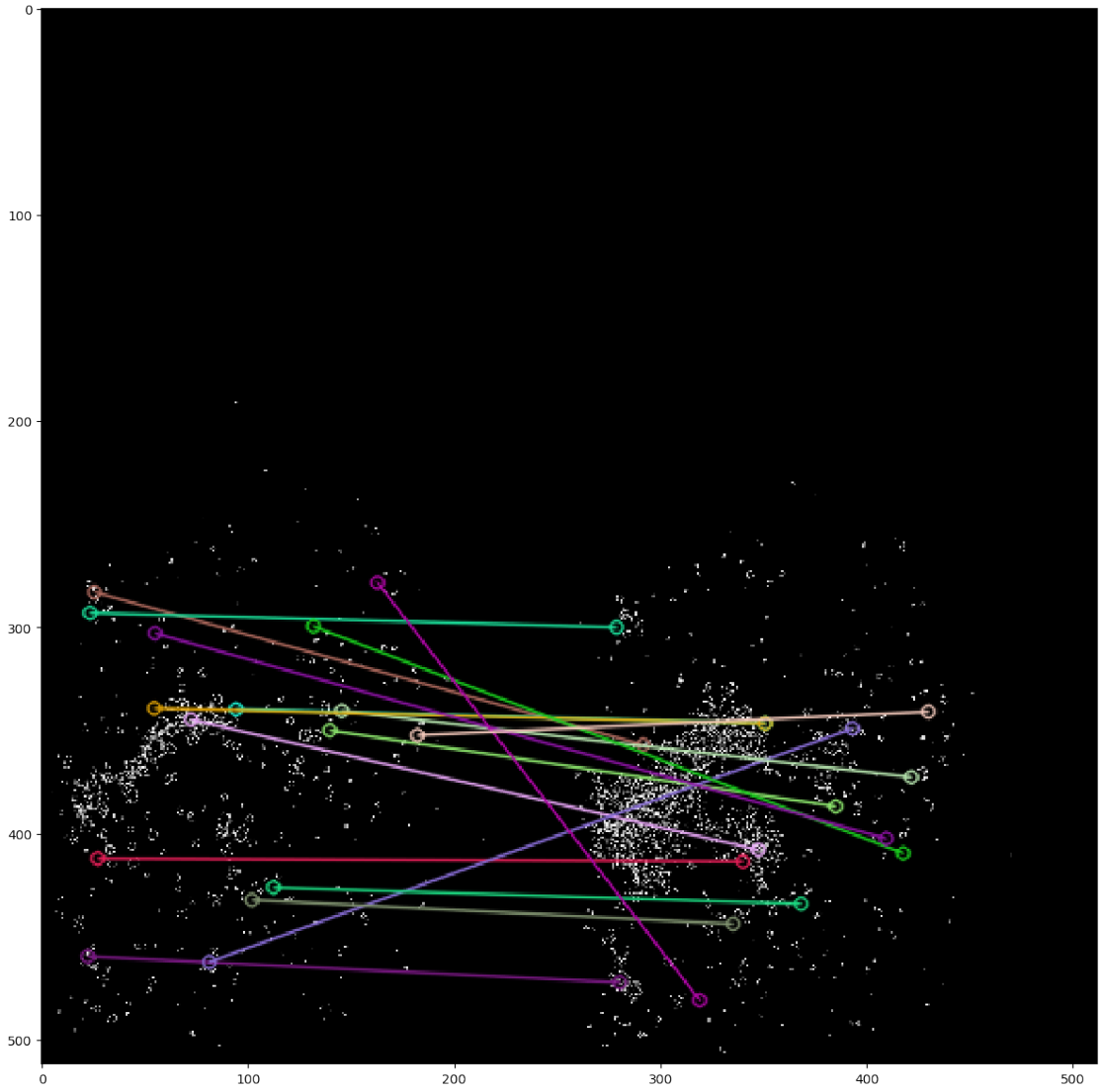
```
    vin = [np.min(src), np.max(src)]
    vout = [0, 65535]
    if tol > 0:
        # Compute in and out limits
        # Histogram
        hist = np.histogram(src,bins=list(range(vin[1] - vin[0])),range=tuple(vin))[0]

        # Cumulative histogram
        cum = hist.copy()
        for i in range(0, vin[1]-vin[0]-1): cum[i] = cum[i - 1] + hist[i]

        # Compute bounds
        total = src.shape[0] * src.shape[1]
        low_bound = total * tol / 100
        upp_bound = total * (100 - tol) / 100
        vin[0] = bisect.bisect_left(cum, low_bound)
        vin[1] = bisect.bisect_left(cum, upp_bound)

    # Stretching
    scale = (vout[1] - vout[0]) / (vin[1] - vin[0])
    vs = src-vin[0]
    vs[src<vin[0]]=0
    vd = vs*scale+0.5 + vout[0]
    vd[vd>vout[1]] = vout[1]
    dst = vd

    return dst.astype(np.uint16)


def im_binarize(img, f):
    temp = img.copy()
    temp[temp<f] = 0
#     temp[temp>=f] = 1
    return temp.astype(np.uint8)

def enhance_blobies(image, f):
    l, r = image[:, :image.shape[1]//2], image[:, image.shape[1]//2:]
    l_adj, r_adj = imadjust(l.copy()), imadjust(r.copy())
    l_bin, r_bin = im_binarize(l_adj, f).astype(np.uint8), im_binarize(r_adj,f).astype
    return l, r, l_bin, r_bin
```

In [3]:
```
# This is a patented technique and not available in all versions of opencv: use `conda
# Open image
image = tiff.imread('tetraspeck.tif')

# default for 16 bits 50000, for 8 bits 200 (=256*50000/64000)
f=50000
```

```
        # left, right, enhanced left and enhanced right image for keypoint detection
        # l, r = image[:, :image.shape[1]//2], image[:, image.shape[1]//2:]
        # l_enh = im_binarize(l, filters.threshold_isodata(r))
        # r_enh = filters.threshold_local(r, 3)

        l, r, l_enh, r_enh = enhance_blobies(image,f)
```

In [4]:
```
        ### Initiate SIFT detector
        # sift = cv2.xfeatures2d.SIFT_create()
        sift = cv2.xfeatures2d.SURF_create()

        # Find the keypoints and descriptors with SIFT
        kp1, des1 = sift.detectAndCompute(l_enh,None)
        kp2, des2 = sift.detectAndCompute(r_enh,None)

        # # BFMatcher with default params
        bf = cv2.BFMatcher()
        matches = bf.knnMatch(des1, des2, k=2)
        # Apply ratio test
        good = []
        for m,n in matches:
            if m.distance < 0.75*n.distance:
                good.append([m])

        print("Keypoints found: {}, {}".format(len(kp1),len(kp2)))
        print("Good matches found: {}".format(len(good)))
        # cv.drawMatchesKnn expects list of lists as matches.
        matches = cv2.drawMatchesKnn(l_enh, kp1, r_enh, kp2, good, outImg=None, flags=2)


        plt.figure(num=None, figsize=(15, 15), dpi=100)
        plt.imshow(matches)
        plt.show()
```

```
Keypoints found: 403, 520
Good matches found: 16
```




```
In [5]: pts1, pts2 = [], []
        for m in good:
            pts1.append(kp1[m[0].queryIdx].pt)
            pts2.append(kp2[m[0].trainIdx].pt)

        pts1 = np.array(pts1).astype(np.float32)
        pts2 = np.array(pts2).astype(np.float32)

        h, mask = cv2.findHomography(pts1, pts2, cv2.RANSAC)
        print(h)
```

```
[[ 1.76228862e-01 -2.82224914e-01  7.04953564e+01]
 [ 6.56672714e-01 -1.74210262e+00  4.97949875e+02]
```

```
[ 1.87485264e-03 -3.73598941e-03  1.00000000e+00]]
```