sift

March 14, 2017

1 SIFT algorithm

SIFT (Scale-Invariant Feature Transform) is an algorithm developped by David Lowe in 1999. It is a worldwide reference for image alignment and object recognition. The robustness of this method enables to detect features at different scales, angles and illumination of a scene.

Silx provides an implementation of SIFT in OpenCL, meaning that it can run on Graphics Processing Units and Central Processing Units as well. This implementation can run on most graphic cards and CPU, making it usable on many setups. OpenCL processes are handled from Python with PyOpenCL, a module to access OpenCL parallel computation API.

Interest points are detected in the image, then data structures called descriptors are built to be characteristic of the scene, so that two different images of the same scene have similar descriptors. They are robust to transformations like translation, rotation, rescaling and illumination change, which make SIFT interesting for image stitching.

In the fist stage, descriptors are computed from the input images. Then, they are compared to determine the geometric transformation to apply in order to align the images.

2 1 - Keypoint extraction

The first step of SIFT is finding features locations ("keypoints") of an image. `silx.sift` provides a `SIFT Plan` to compute these keypoints from an image. Keypoints are characterized by * A location in the image (x, y) * A *scale* (is the feature visible from a distance or only when getting closer) * A characteristic angle of the feature * A *descriptor*, which is a data structure containing compressed information on the keypoint

To visualize these keypoints, we are going to compute them from an image.

```
In [1]: %pylab inline
```

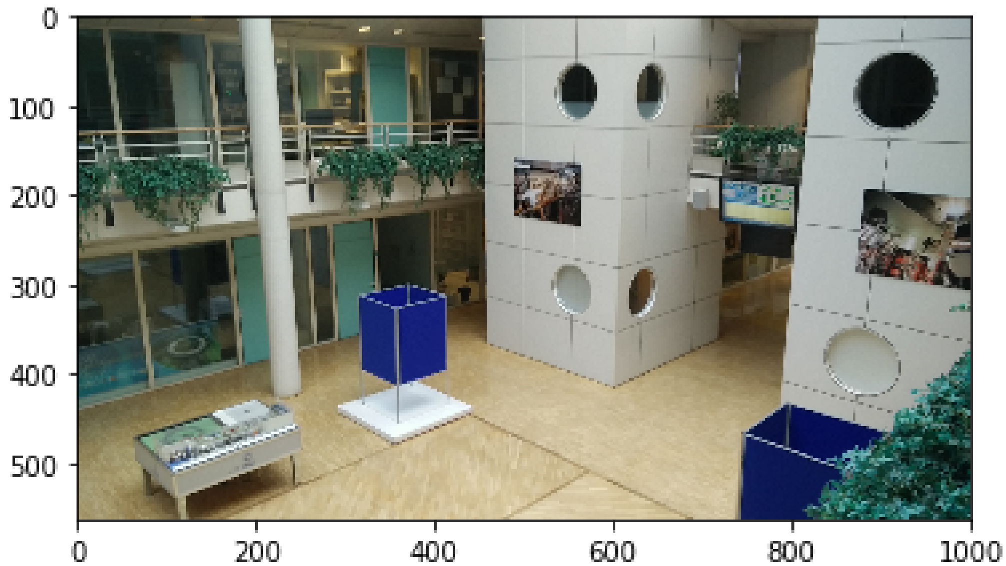
```
Populating the interactive namespace from numpy and matplotlib
```

Open a first image and find its characteristic keypoints.

```
In [2]: import fabio
        image1 = fabio.open("IMG_20170309_083429.tiff").data
        imshow(image1)
        print(image1.shape, image1.dtype)   # RGB image: 563x1000x3 dtype:uint8
```

```
WARNING:tifimage:Third dimension is the color
```

```
((563, 1000, 3), dtype('uint8'))
```



In the following code, replace "CPU" with "GPU" to test parallel computing on your graphics card.

```
In [3]: from silx.image import sift
        sift_plan = sift.SiftPlan(image1.shape,
                                  image1.dtype,
                                  devicetype="CPU")

        # Alternative shortcut using template keyword:
        # sift_plan = sift.SiftPlan(template=image1,
        #                            devicetype="GPU")
```

```
WARNING:sift.plan:Unable to find suitable device, selecting device: 0,0
```

Use the SIFT plan to find keypoints in an image.

```
In [4]: # Find keypoints
        %time keypoints = sift_plan(image1)

        # print information about detected keypoints
        print("Number of keypoints: %s" % len(keypoints))
        print("Last keypoint's content:")
```

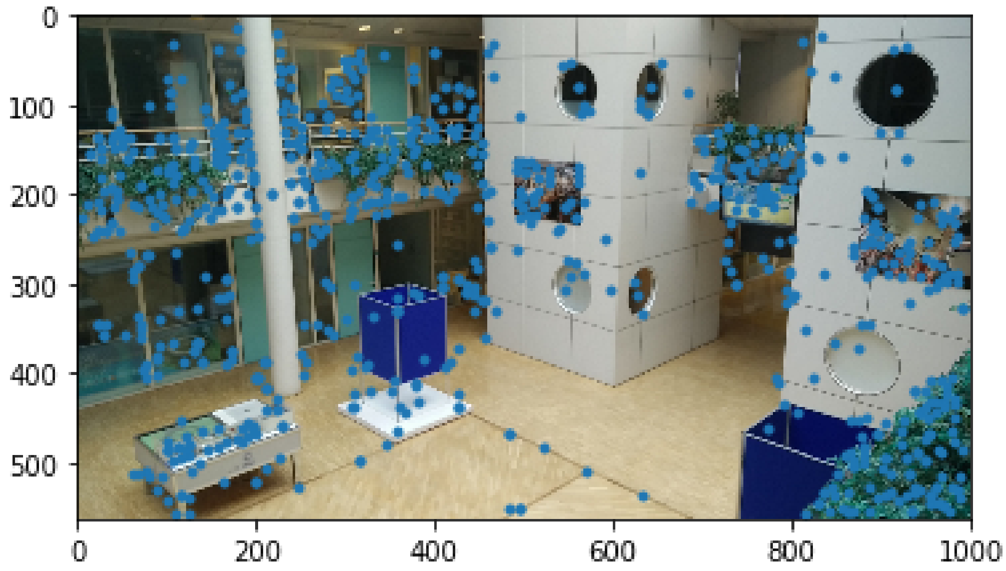
```
        print(keypoints.dtype)
        print("x: %.3f \t y: %.3f \t scale: %.3f \t angle: %.3f" %
              (keypoints[-1].x,
               keypoints[-1].y,
               keypoints[-1].scale,
               keypoints[-1].angle))
        print("descriptor:")
        print(keypoints[-1].desc)
```

```
CPU times: user 528 ms, sys: 4 ms, total: 532 ms
Wall time: 531 ms
Number of keypoints: 867
Last keypoint's content:
(numpy.record, [('x', '<f4'), ('y', '<f4'), ('scale', '<f4'), ('angle', '<f4'), ('c
x: 546.704          y: 307.066          scale: 57.176          angle: 0.422
descriptor:
[ 37   0   0   5   3   0   0  82  24   0   0  23  69   5  16 103  31   1
   0  59 110   2   1  18  38   3   6   8   9   2   1  17  44   1   1  39
  45   2   2  89 123   8   3  11  25   1   5 123  33   6  25 123 117   2
   0  17  51   4  18  31  17  36  32  38  31  64  52  32  56   5   7  24
 123 123  41   7   3   0   0  26  33  56  82 123  14   0   0   0   2   0
  18 113  87  41  19   3  75 117  32   1   0   0   0   8  26 123  27   0
   0   0   0   0   1  26   9  10   1   0   0   0   0   0   0  11   5   0
   0   0]
```

Display keypoints on the image:

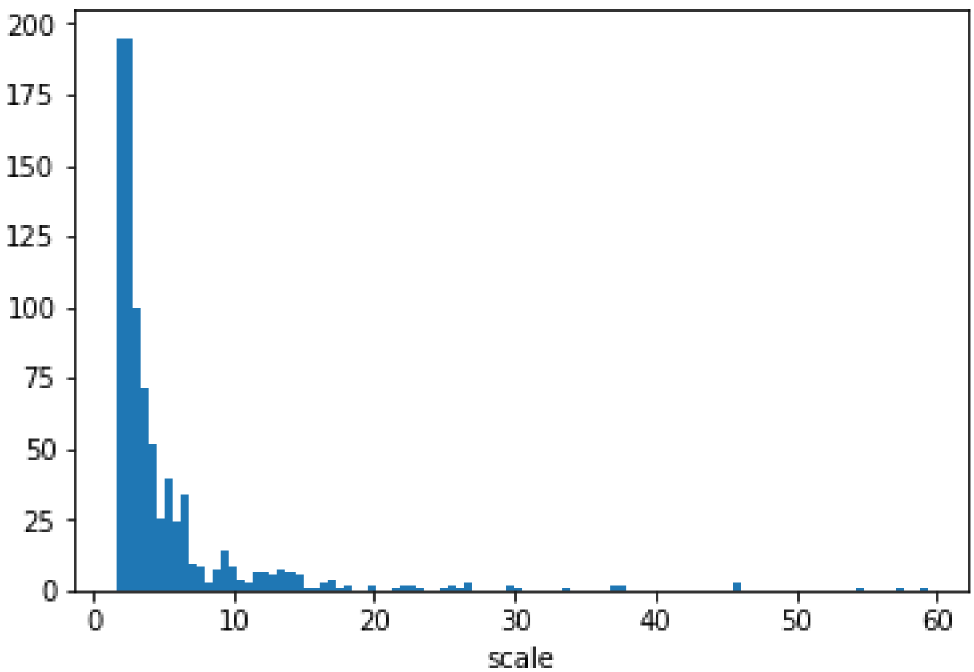
```
In [5]: imshow(image1)
        plot(keypoints[:].x, keypoints[:].y,".")
```

```
Out[5]: [<matplotlib.lines.Line2D at 0x7fd068048450>]
```



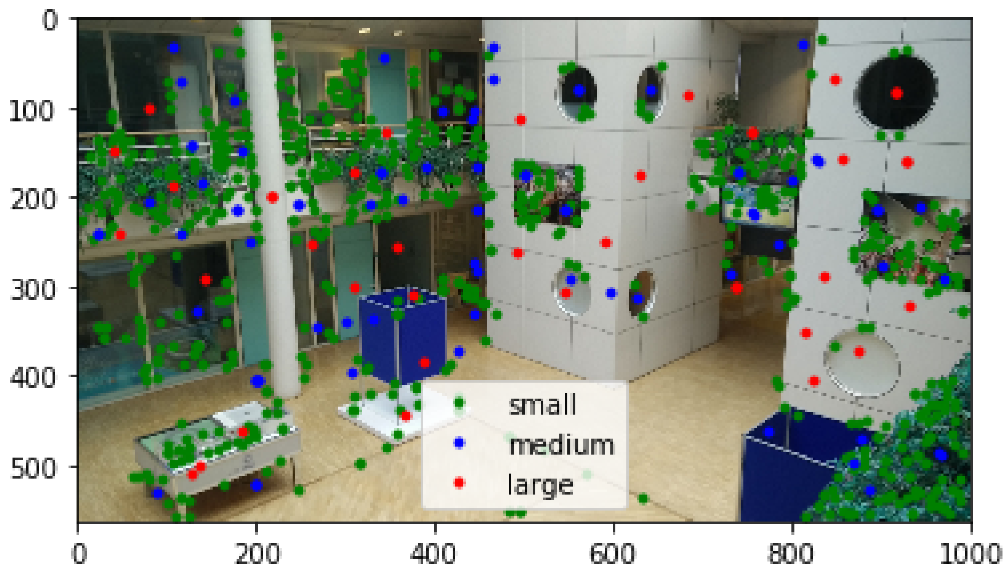
```
In [6]: # Diplaying keypoints histogram by scale:
        hist(keypoints[:].scale, 100)
        xlabel("scale")
```

```
Out[6]: <matplotlib.text.Text at 0x7fd0541d91d0>
```




```
In [7]: # One can see 3 groups of keypoints, boundaries at: 8 and 16.
        # Let's display them in different colors.
        S = 8
        L = 16
        small = keypoints[keypoints[:].scale < S]
        medium = keypoints[numpy.logical_and(keypoints[:].scale < L,
                                             keypoints[:].scale >= S)]
        bigger = keypoints[keypoints[:].scale >= L]
        imshow(image1, cmap="gray")
        plot(small[:].x, small[:].y,".g", label="small")
        plot(medium[:].x, medium[:].y,".b", label="medium")
        plot(bigger[:].x, bigger[:].y,".r", label="large")
        legend()
```

```
Out[7]: <matplotlib.legend.Legend at 0x7fd06c323e90>
```



3 2 - Keypoint matching

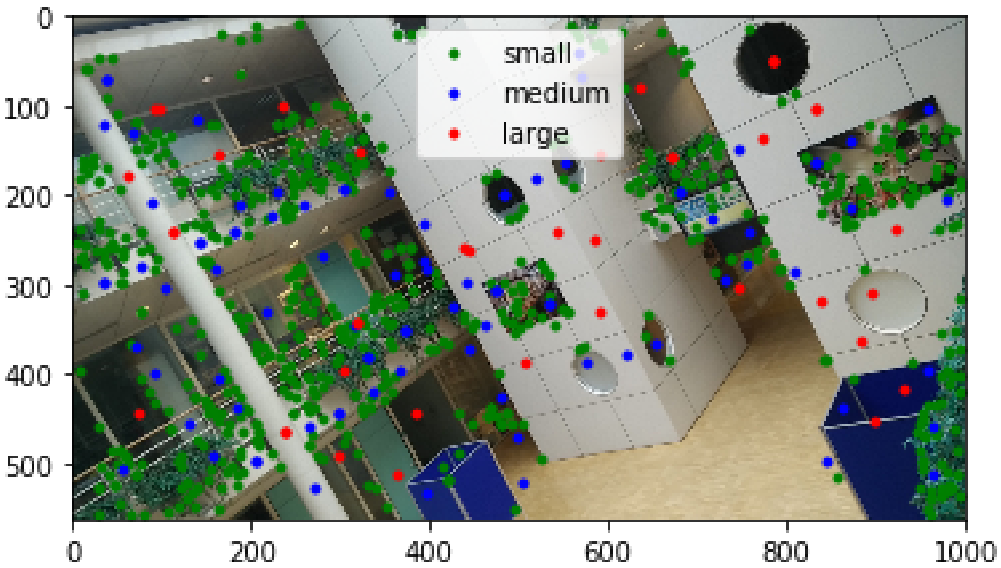
The second step of SIFT image alignment is to *match* the keypoints computed from two images. If there are enough matching pairs of keypoints, we will be able to infer a transformation mapping a set of keypoints to the other, and thus apply the inverse transformation to align the images.

Use your previous SiftPlan to compute keypoints for the second image.

```
In [8]: image2 = fabio.open("IMG_20170309_083439.tiff").data
        imshow(image2)

        # use the same plan to compute keypoints
        # (both images have same size and dtype)
        keypoints2 = sift_plan(image2)



        # same procedure as previously: show keypoints on image by scale
        small2 = keypoints2[keypoints2[:].scale < S]
        medium2 = keypoints2[numpy.logical_and(keypoints2[:].scale < L,
                                               keypoints2[:].scale >= S)]
        bigger2 = keypoints2[keypoints2[:].scale >= L]
        plot(small2[:].x, small2[:].y,".g", label="small")
        plot(medium2[:].x, medium2[:].y,".b", label="medium")
        plot(bigger2[:].x, bigger2[:].y,".r", label="large")
        legend()
```

```
WARNING:tifimage:Third dimension is the color
```

```
Out[8]: <matplotlib.legend.Legend at 0x7fd06c25dd90>
```



Then we can use MatchPlan to find the offset between the two images. The following calculation assumes that images are just translated, not rotated (our example is not ideal).

```
In [9]: mp = sift.MatchPlan()
        match = mp(keypoints, keypoints2)
        print("Number of Keypoints with for image 1 : %i" % keypoints.size)
        print("For image 2 : %i" % keypoints2.size)
        print("Matching keypoints: %i" % match.shape[0])
        print(match.dtype)
        print(match.shape)

        from numpy import median
        print("Measured offsets dx: %.3f, dy: %.3f" %
              (median(match[:,1].x - match[:,0].x),
               median(match[:,1].y - match[:,0].y)))
```

```
Number of Keypoints with for image 1 : 867
For image 2 : 826
Matching keypoints: 321
(numpy.record, [('x', '<f4'), ('y', '<f4'), ('scale', '<f4'), ('angle', '<f4'), ('d
(321, 2)
Measured offsets dx: -14.709, dy: 179.898
```

4 3 - Image alignement

Given two sets of keypoints computed from two images `image1` and `image2`, we determine the transformation mapping one set of keypoints to the other set of keypoints.

(1)Keypoints computation	image1 $\longrightarrow K_1$ image2 $\longrightarrow K_2$	S = SiftPlan(template=image1); K1 = S(image1) K2 = S(image2)
(2)Keypoints matching	$K_1 \leftrightarrow K_2$	M = MatchPlan(); M(K1, K2)
(3)Image alignment	$\begin{cases} \mathcal{T} : K_1 \mapsto K_2 \\ \text{image2\_al} = \mathcal{T}^{-1}\,(\text{image2}) \end{cases}$	Transformation inference Apply transformation to align image2 wrt image1

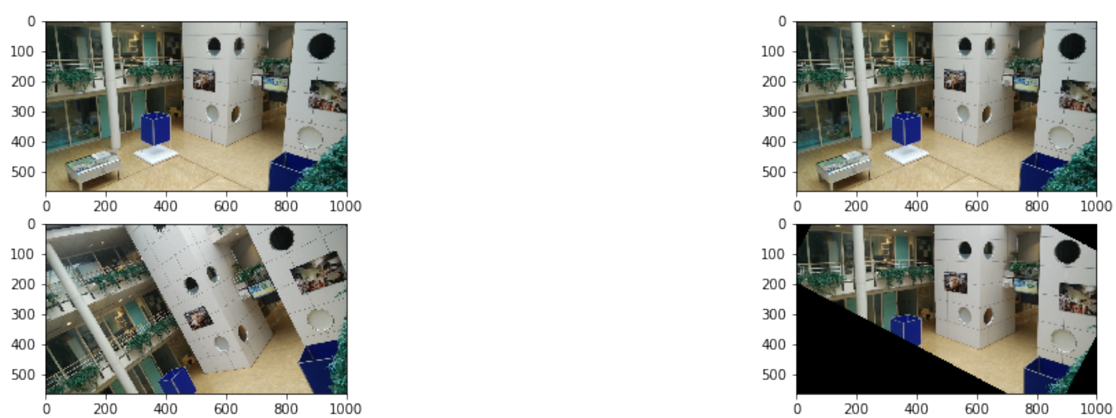
ImageAlignement

The `LinearAlign` class wraps up all the steps of the SIFT alignment, from keypoints computation to transformation inference. Use it to align images that underwent linear geometric transformations (translation, rotation, scaling, shear).
All you have to do is

```
L = LinearAlign(image1) # prepare the alignment plan for image 1 size, dtype, ...
image2_aligned = L(image2) # determine the transformation mapping image1 to image2
```

** Align image2 with image1, using a translation and a rotation: **

```
In [12]: sa = sift.LinearAlign(image1, devicetype="GPU") # choose either "CPU" or "
         image2_aligned = sa.align(image2)

         # plot images side by side for visual comparison
         figure(figsize=(18,5))
         subplot(2,2,1)
         imshow(image1)
         subplot(2,2,2)
         imshow(image1)
         subplot(2,2,3)
         imshow(image2)
         subplot(2,2,4)
         imshow(image2_aligned)

Out[12]: <matplotlib.image.AxesImage at 0x7fd0315e0ed0>
```



In [ ]: