AFid in MATLAB

This example describes how MATLAB can be used identify autofluorescence from two channel microscopy images using k-means clustering. This requires the functions `afIdentifier.m` and `glowIdentifier.m` to be in the working directory.

Step 1: Read in images

The two images from which autofluorescence is to be identified is read into MATLAB.

```
ch1 = imread('1-1.tif');
ch2 = imread('1-2.tif');
```

Step 2: Obtain a mask of objects to analyse

A mask of objects can be provided (e.g. generated in ImageJ), or generated by thresholding within MATLAB.

Step 2a: Provide an already generated mask

Here, the mask was generated in ImageJ by Gaussian blurring both images, performing a Niblack threshold on both channels, and performing the binary AND operation.

```
mask = logical(imread('mask.tif'));
```

Step 2b: Generate a mask within MATLAB

Here, the mask was generated within MATLAB applying the `imbinarize` function on the Gaussian blurred images of both channels, employing Otsu's threshold using the `graythresh` function. An intersection mask of both threshold masks was then generated.

```
sig = 2;
ch1Blurred = imgaussfilt(ch1,sig);
ch2Blurred = imgaussfilt(ch2,sig);

ch1Threshold = imbinarize(ch1Blurred,graythresh(ch1Blurred));
ch2Threshold = imbinarize(ch2Blurred,graythresh(ch2Blurred));

mask = ch1Threshold & ch2Threshold;
```

Once a mask is generated, a size filter can be used to exclude objects that are too small or too large.

```
minArea = 20;
maxArea = 10000;

mask = bwareafilt(mask,[minArea maxArea]);
```

Step 3: Autofluorescence Classification

Here, objects in the intersection mask are classified as autofluorescence or non-autofluorescence based on a correlation cut-off, the results of *k*-means clustering, or both. The pixel correlation of objects in the intersection mask across the two provided image channels are measured. Additional texture measurements are also made, namely the standard deviation and kurtosis measurements of the objects in the two channels, which serve as input parameters for *k*-means classification. In MATLAB this performed using the `afIdentifier` function.

Step 4a: Classifying autofluorescence using pixel correlation

In the simplest case, this is performed simply with correlation, classifying objects in the mask as autofluorescence if the two-channel pixel correlation is found to be greater than a given cut-off, here set to 0.60:

```
[maskAF, ch1AFRemoved, ch2AFRemoved] = afIdentifier(ch1, ch2, mask, 
'Corr', 0.60);
```

Here, `ch1` and `ch2` are the two images read in in Step 1, and `mask` is a mask of objects to be measured as generated in Step 2.

The pair-argument value `'Corr'` represents a correlation cut-off set between -1 and 1.

The outputs are `maskAF`, a mask of all autofluorescent objects identified, and `ch1AFRemoved` and `ch2AFRemoved`, the initial images with all autofluorescent objects reset to 0.

Step 4b-i: Classifying autofluorescence using *k*-means clustering with manual k

If *k*-means is to be performed to classify autofluorescent objects with a given *k*, `afIdentifier` can be used as:

```
[maskAF, ch1AFRemoved, ch2AFRemoved] = afIdentifier(ch1, ch2, mask, 
'k', 6);
```

Here, the pair-argument value `'k'` is used to set the number of clusters to a positive integer, set to 6 in this example.

Step 4b-ii: Classifying autofluorescence using *k*-means clustering with estimated *k*

If *k* is to be estimated, `afIdentifier` can be used as:

```
[maskAF, ch1AFRemoved, ch2AFRemoved, kBest] = afIdentifier(ch1, ch2, 
mask,  'kAuto', 1, 'k', 20);
```

Here, the pair-argument value `'kAuto'` set to 1 results in *k* being estimated rather than user-provided. An elbow test and the t-statistic are used to estimate an optimal value of *k* between 3 and the value set by the user, now using the pair-argument value `'k'` to set the maximum *k* to 20.

Step 4b-iii: Classifying autofluorescence using *k*-means clustering and correlation

Finally, *k*-means can be combined with correlation using the pair-argument value `'Corr'` to specify a correlation cut-off for objects within the identified autofluorescence cluster:

```
[maskAF, ch1AFRemoved, ch2AFRemoved] = afIdentifier(ch1, ch2, mask,
'k', 6, 'Corr', 0.60);
```

```
[maskAF, ch1AFRemoved, ch2AFRemoved, kBest] = afIdentifier(ch1, ch2,
mask,  'kAuto', 1, 'k', 20, 'Corr', 0.60);
```

Step 4: Glow Removal

The full body of objects are not always captured by the thresholds used, and so further ‘glow removal’ is to be performed. This is performed by distributing expansion points within autofluorescent objects and drawing lines outwards to capture the boundary of the objects. A boundary is identified when the pixel intensity along the line starts to increase as defined by the blurred image. In MATLAB this performed using the `glowIdentifier` function.

This function is used as:

```
[glow1, glow2, ch1GlowRemoved, ch2GlowRemoved] = glowIdentifier(ch1, ch2,
maskAF, 'TraceSensitivity', 20, 'Sigma', 2);
```

Here, `ch1` and `ch2` are our original images, and `maskAF` is the mask of autofluorescent objects.

The pair-argument value `'TraceSensitivity'` represents the number of steps taken by a tracing algorithm when dropping expansion points within an object. The lower this parameter, the closer these expansion points.

The pair-argument value `'Sigma'` represents any Gaussian blurring done to the images during the boundary-identification process. This ensures that the boundaries are not misidentified as a result of suddenly increasing pixel values. Setting this parameter to 0 (default) results in no Gaussian blurring being performed.

The outputs are `glow2` and `glow2`, masks of the glow identified, and `ch1GlowRemoved` and `ch2GlowRemoved`, the initial images with all identified glow reset to 0.