Add an own function to Microscopy Image Browser

(*general case without GUI*)

Ilya Belevich and Darshan Kumar

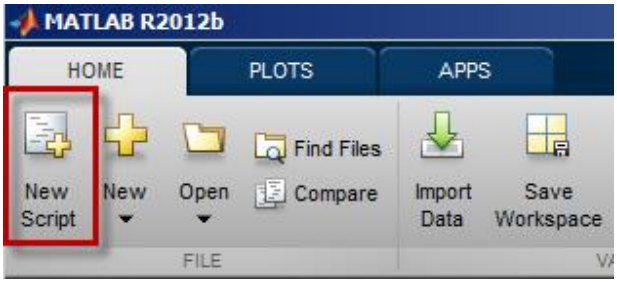
This tutorial will demonstrate how to write a simple filtering function and connect it to Microscopy Image Browser (im_browser) as a plugin.

Task: write a function of automatic global image thresholding by Otsu's method and generate the *Mask* layer based on results of thresholding.

Reference: the files described in this tutorial located in directory:

```
im_browser\Plugins\Tutorials\otsuThresholding\
```

Step 1. Start a new script (*Matlab->Home->New script button*)



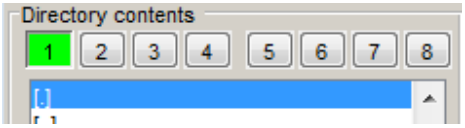
Step 2. Write a function name

```
function handles = otsuThresholding(par1, par2, h_im_browser)
% par1, par2 will not be used
% h_im_browser - is a handle to the main program (im_browser)

% get the handles structure of the main program
% using this structure it is possible to obtain status of widgets within MIB
GUI:
%    for example the following command returns the status of the show mask
%    checkbox: 1-checked, 0-unchecked:
%        value = get(handles.h.maskShowCheck,'value');
% and also get access to the classes that have dataset (see below)
handles.h = guidata(h_im_browser);
```

From this moment, it is possible to get access to all functions of the im_browser classes. See more in the description of classes inside MID: *im_browser->Menu->Help->Class reference*, or on the website: http://mib.helsinki.fi/help/api/index.html

There following classes are available:



- *handles.h.Img{handles.h.Id}.I* – the main class where the images are stored; the variable `handles.h.Id` has index (from 1 to 8) of the currently shown dataset (defined by the buffer buttons in the Directory Contents panel). http://mib.helsinki.fi/help/api/classimage_data.html
- *handles.h.Img{handles.h.Id}.I.hROI* – a subclass with information about regions of interest defined in the ROI panel. http://mib.helsinki.fi/help/api/classroi_region.html
- *handles.h.Img{handles.h.Id}.I.hLabels* - a subclass for keeping information about text annotations. http://mib.helsinki.fi/help/api/class_labels.html
- *handles.h.Img{handles.h.Id}.I.hMeasure* - a subclass for keeping information manual length measurements (Menu->Tools->Measure length->Measure Tool); http://mib.helsinki.fi/help/api/class_measure.html
- *handles.h.U* - class with information of the undo system. There is no need to access this class directly because the backup/undo process is regulated using the `ib_do_backup/ib_do_undo` functions. http://mib.helsinki.fi/help/api/classimage_undo.html

The access to the currently selected image done using
`handles.h.Img{handles.h.Id}.I.[functionName]`
syntax, where `handles.h.Id` contains the index of the opened dataset from 1 to 8.
For example,

1. get width of the image:
```
Width = handles.h.Img{handles.h.Id}.I.width;
```
2. redraw image in the Image View panel of MIB
```
handles.h.Img{handles.h.Id}.I.plotImage(handles.h.imageAxes, handles.h, 0);
```

Step 3. Obtain number of color channels in the opened image

The Otsu thresholding is done on a single channel. Let's check how many color channels are displayed at the moment

```
% get number of displayed color channels
% it is stored in the slices variables of
% handles.h.Img{handles.h.Id}.I class
col_channels = handles.h.Img{handles.h.Id}.I.slices{3};
if numel(col_channels) ~= 1;
    msgbox(sprintf('Warning!\nYou need to select a color channel for the
thresholding.\nPlease keep only one channel in the View Settings Panel-
>Colors...'),'Wrong number of color channels!','Error');
    return;
end
```

Step 4. Let's ask whether to threshold a single slice or the whole dataset

```
% ask whether to threshold a single slice or a whole dataset
button = questdlg('Would you like to threshold a whole dataset or a single
slice?','Otsu thresholding','Whole dataset','Single slice','Cancel','Whole
dataset');
if strcmp(button, 'Cancel'); return; end;   % cancel and return
```

Step 5. Allocate space for the new mask or store the existing mask

The results of the thresholding will be stored in the *Mask* layer. Allocate space for the *Mask* layer if it is not present; otherwise store the existing Mask so that it can be restored with Ctrl+Z shortcut

```
% preallocate space for the mask if it is not present
% maskExist variable keep information whether mask exist (1) or not (0)
if handles.h.Img{handles.h.Id}.I.maskExist == 0
    % allocate space and set maskExist switch to 1
    handles.h.Img{handles.h.Id}.I.clearMask(0);
else
    % store the existing Mask, so that the thresholding may be undone with
Ctrl+Z
    handles.h = ib_do_backup(handles.h, 'mask', 1);
end
```

There are several options to make backup:

1. Make backup of the complete 3D dataset for the Mask layer
   ```
   handles.h = ib_do_backup(handles.h, 'mask', 1);
   ```
2. Make backup of the shown 2D slice for the Selection layer
   ```
   handles.h = ib_do_backup(handles.h, 'selection', 0);
   ```
3. Make backup of subarea of the dataset for the Image layer
   ```
   Options.x = [50, 150];
   Options.y = [50, 150];
   Options.z = [5, 10];
   handles.h = ib_do_backup(handles.h, 'image', 1, Options);
   ```

Step 6. Threshold the currently shown slice

For historical reasons MIB has many methods that duplicate each other. In general, it is best to use the wrapper functions such as `ib_getSlice/ib_setSlice, ib_getStack/ib_setStack, ib_getDataset/ib_setDataset` to work with the datasets.

When these wrapper functions are used the returned dataset (or datasets, when the ROI mode is enabled) is assigned to a cell array. It is important that the returned images will be cropped depending on settings of main MIB GUI:

1. When the block mode is enabled  the returned dataset is limited only to the area seen on the screen. It is possible to override this value in the options structure for the functions (`options.blockModeSwitch`)
2. When the ROI mode is enabled  the returned datasets are limited to the selected ROIs

For example, get the currently shown image

```
img = ib_getSlice('image', handles.h);
```

img{1} – is the image, or the image that belongs to the 1st shown ROI; img{2} – for the 2nd shown ROI.

Alternatively, there are number of ImageData class methods that can be used:

`getData2D/setData2D, getData3D/setData3D, getData4D/setData4D, getSlice/setSlice, getFullSlice/setFullSlice, getDataset/setDataset…`

They can be called like this and the resilting images are not cell arrays, but just matrices.

```
im = handles.h.Img{handles.h.Id}.I.getData2D('image');
```

```
if strcmp(button, 'Single slice')
    img = ib_getSlice('image', handles.h);
    % the 'image' parameter defines that the image is required, alternatively
    % it can be: 'model', 'mask', or 'selection'
    % when other parameters omitted the function call will take the
    % currently shown slice. The function returns a cell with image or
    % an array with cells when several ROIs present.

    % do the Otsu thresholding,
    % the for-loop is needed because the image is return as a cell-array.
    % When ROIs are shown number of elements in the array is equial to
    % number of ROIs. When ROIs are not present the cell-array has only one
    % element.
    for roi=1:numel(img)
        level = graythresh(img{roi});
        % use the same variable to store the thresholded image
        img{roi} = im2bw(img{roi}, level);
    end

    % update the Mask layer with a new mask
    ib_setSlice('mask', img, handles.h);
end
```

Step 7.Threshold the whole dataset

Two methods are demonstrated; pick one method and comment with % another.

Before start lets initialize the waitbar to show the progress of the thresholding

```
% since it may take time, lets add a waitbar to show the progress of the
% operation
wb = waitbat(0, 'Please wait...','Name','Otsu thresholding');
% calculate the number of sections to process
maxIndex = ...
  handles.h.Img{handles.h.Id}.I.time*handles.h.Img{handles.h.Id}.I.no_stacks;
% create a counter
counter = 1;
```

Method A: threshold complete 3D stacks

```
if strcmp(button, 'Whole dataset')
    % do the loop via the time dimension
    % if you never plan to work with the time dimension this loop
    % can be omitted
    for t=1:handles.h.Img{handles.h.Id}.I.time
        % get 3D stack
        img = ib_getStack('image', handles.h, t);

        % loop across number of ROI,
        % when ROIs are not shown the loop is one element only
        for roi=1:numel(img)
            for slice = 1:size(img{roi},4)
                % do the Otsu thresholding
                level = graythresh(img{roi}(:,:,:,slice));
                % use the same variable to store the thresholded image
                img{roi}(:,:,:,slice) = im2bw(img{roi}(:,:,:,slice),level);
                waitbar(counter/maxIndex, wb);  % update the waitbar
                counter = counter + 1;  % increase the counter
            end

            % at the end we should squeeze dataset, because dimensions of
            % the mask are [height, width, depth, time]
            % and convert it to 8bit (because you might threshold the
            % 16bit image
            img{roi} = squeeze(uint8(img{roi}));
        end

        % return stack to the Mask layer
        ib_setStack('mask', img, handles.h, t);
    end
delete(wb);     % delete the waitbar
end
```

Method B: threshold dataset slice-by-slice, it is slower, but more memory friendly

```
if strcmp(button, 'Whole dataset')
    % do the loop via the time dimension
    % if you never plan to work with the time dimension this loop
    % can be omitted
    for t=1:handles.h.Img{handles.h.Id}.I.time
       % define orientation to use: 1, 2 or 4 (xy)
        orientation = 4;

        % define the time point of the Z-stack, it is not needed when when
        % the t-loop is omitted
        options.t = [t t];

        % make a loop across slices of Z-stacks
        for slice = 1:handles.h.Img{handles.h.Id}.I.no_stacks
            % get slice
            img = ib_getSlice(...
                    'image', handles.h, slice, orientation, NaN, options);

            for roi=1:numel(img)
                % do the Otsu thresholding
                level = graythresh(img{roi});
                img{roi} = im2bw(img{roi}, level);
            end

            ib_setSlice(...
                  'mask', img, handles.h, slice, orientation, NaN, options);

            waitbar(counter/maxIndex, wb);  % update the waitbar
            counter = counter + 1;          % increase the counter
        end
    end
delete(wb);     % delete the waitbar
end
```

Step 8. Set on the Show Mask check box to the checked state

By default, the Show mask checkbox is unchecked so the Mask layer is not displayed, here we change its state to see the mask layer.

Handles of the GUI widgets can be easily checked using `GUIDE` of Matlab:

1. type ‘guide’ in the Matlab command window;
2. open `im_browser.fig`
3. double click on any widgets check in the Inspector window the Tag parameter

```
set(handles.h.maskShowCheck,'value',1);
```

Step 9. Update widgets of MIB

Not really needed for this case, but it may be required in other cases to update widgets of the im_browser.

```
% update widgets in the im_browser GUI
handles.h = updateGuiWidgets(handles.h);
```

Step 10. Redraw the image in the Image Panel of MIB

```
% redraw the im_browser
handles.h = ...
    handles.h.Img{handles.h.Id}.I.plotImage(handles.h.imageAxes, handles.h, 0);
```

Step 11. Save function

Save function to the `Plugins\Tutorials\otsuThresholding` directory within the `im_browser` directory (for example, the full path may look like this: `C:\Scripts\im_browser\Plugins\ Tutorials\otsuThresholding\otsuThresholding.m`)
It is important that the directory name should be the same as the name of the function: *otsuThresholding* in this example.

Step 12. Re-start **`im_browser`** to connect the plugin.

Now the thresholding function may be started from the menu:
`Menu->Plugins->Tutorials-> otsuThresholding`