Calibrating a microscope using a calibration target

Hernan G. Garcia

April 3, 2011

1 Introduction

When we take an image using a CCD all distances are given in pixels, the fundamental spatial unit of a digital camera. Of course, we can guess the size of what we're looking at from the supposed magnification of the scope (the magnification of the objective that was chosen). This tutorial will walk you through calibrating a microscope such that you can include a scale bar on all of your images based explicitly on using a known length standard. Unfortunately, even though widespread, this is not common practice in all biology. You'll often find images in textbooks (and even research papers) without any scale bars.

2 Taking a good image

You will take an image of your calibration target using Micro-Manager to acquire and save the image (refer to the Micro-Manager manual and the TAs!). The calibration target is a cross with lines spaced every 10 μm. Notice that the targets we have already come with a coverslip on top of the slide. We will always image through the coverslip. For 100x imaging we'll have to put oil on it. When you're done imaging make sure you wipe the oil off using a cotton swab and isopropanol. You can find both of these cleaning supplies in the drawers underneath the microscopes.

Fig. 2.1 shows two representative images of the same calibration target. Both images are good enough for the purposes of calibration. There is a clear aesthetic difference between the two images. However, the one that looks "nicer" is a little bit crooked. The image on the left presents some "bubbles" which could be due to the coverslip being dirty.

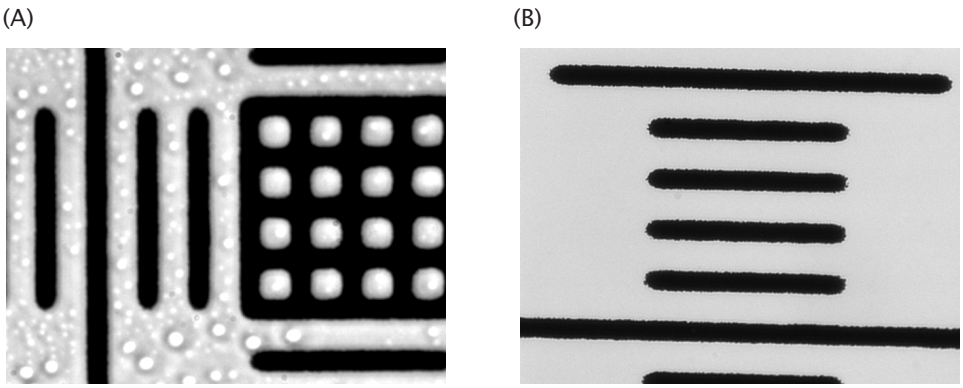

Figure 2.1: Brighfield images of a calibration target at 100x magnification. Both images are acceptable for calibration purposes. (B) just looks a little bit nicer than (A), probably because the coverslip in (A) wasn't cleaned properly. However, (A) is clearly a little bit crooked.

3 Loading the images in Matlab

As you have seen already in the microscopy tutorial, images can come in different bit depths. Most of the digital cameras you will use throughout the course have a native bit depth of 16 bits. This means that Windows won't be able to display them using the standard viewers. However, ImageJ and Matlab won't have any problems.

We start in Matlab by going to the folder where you have your image. This can be done by clicking on the "..." to the right of the white bar and just browsing for the folder. First, we load the image using the command `imread` and put it in the variable `Im`

```
Im = imread('100xTarget.tif');
```

Now, remember that a 16 bit image has $2^{16} = 65536$ levels of grey, but that your screen has only $2^8 = 256$ levels. If we want to display the image on the screen we'll have to tell Matlab how to scale the image down to the screen bit depth. For example, if you type

```
imshow(Im);
```

you'll just see a black image. The command `imshow` has a way of inputting the bit depth scaling. One option, for example, is to do

```
imshow(Im,[]);
```

This tells Matlab to grab the brightest pixel in the image and assign it a brightness of 1 and to grab the darkest one and assign it a brightness of 0. Matlab uses the $[0, 1]$ range for displaying images. Alternatively, one can specify the scaling by giving the thresholds between the `[]` in `imshow`. Please, refer to the help in order to learn a little bit more about it.

4 Measuring distances and calibrating

Now that we have successfully displayed the image we want to measure distances. We could write a really fancy algorithm for automatically finding the position of the bars. However, sometimes it's easier to do things manually. Matlab has a function called `ginput`, which allows you to click on different parts of the image and which returns the coordinates of where the clicks where made. We ask for two clicks and put the information in the variable `Pos`

```
Pos = ginput(2);
```

An example of where to click is shown in fig. 4.1. You want to click on the two bars that are the farthest apart from each other. This should decrease the error in finding the position of the edge of each mark.

The variable `Pos` is a matrix. Each row corresponds to a click of the mouse. The first column gives the x-coordinate and the second column gives the y-coordinate. Calculating the distance between the two points is just a matter of geometry

```
sqrt((Pos(1,1)-Pos(2,1))^2+(Pos(1,2)-Pos(2,2))^2);
```

In this particular case we get 928 pixels. This corresponds to six divisions or 60 μm. Therefore, the calibration factor is 0.065 μm/pixel. Now you can go back to your images and do a sanity check. For example, given this calibration factor, do the images you took of *E. coli* make sense?

5 Adding a scale bar

Adding a scale bar is relatively easy. One approach is to just set some pixels to 1, which is white. In the case of our image, its size in pixels is 1024x1344. This can be found out using the command

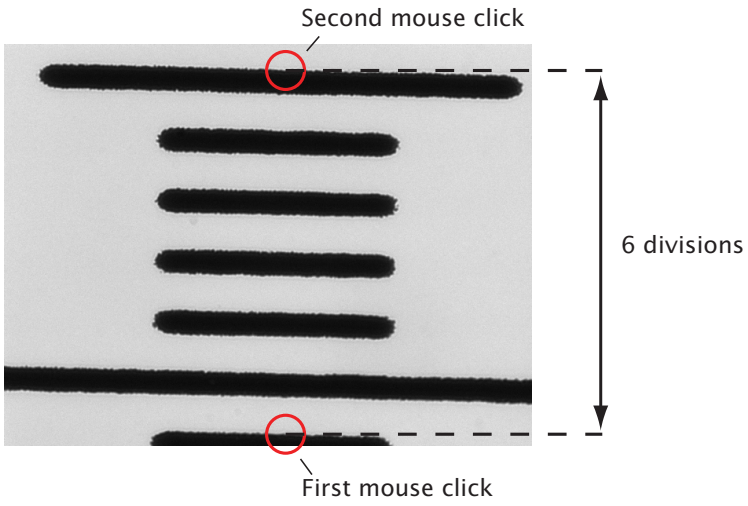

Figure 4.1: Using `ginput` to measure the distance between marks on the calibration slide. `ginput` is used to click on two different marks. The function outputs the coordinates of the clicks allowing us to compare distances in pixels with distances in μm.

```
size(Im)
```

This command gives the size of the matrix with the image. The first number corresponds to the number of rows (y-coordinate) and the second one is the number of columns (x-coordinate). Let's make a 10 μm scale bar, which corresponds to 154 pixels. First, we copy the image to a new variable. We also use the command `mat2gray` to convert the image directly to the $[0, 1]$ range

```
ImScale=mat2gray(Im);
```

Notice that if you do

```
imshow(ImScale)
```

you don't need to add the `[]` as an option of `imshow`. This is because `mat2gray` has already rescaled the image for the $[0, 1]$ range.

Now, let's make a bar that is 154 pixels wide and 10 pixels in height and locate it on the lower-right edge of the image

```
ImScale(950:959,1100:1253)=ones(10,154);
imshow(ImScale)
```

Finally, we can save our newly generated image to a new TIF file

```
imwrite(ImScale,'ImageScalebar.TIF','TIF');
```

ImageJ can also add scale bars to images. This can be done by loading the image and then going to `Analyze` $\rightarrow$ `Tools` $\rightarrow$ `Scale Bar...`. From there everything is pretty self-explanatory.