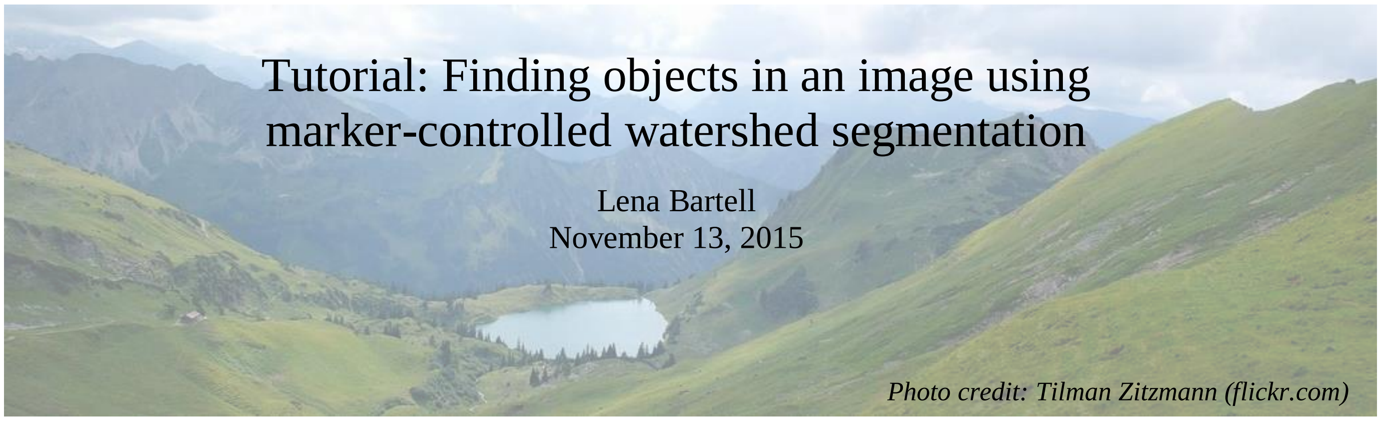


Tutorial: Finding objects in an image using marker-controlled watershed segmentation

Lena Bartell
November 13, 2015

Photo credit: Tilman Zitzmann (flickr.com)

A few notes before we begin

General image-analysis process

The most important parts of any image-analysis technique are often ignored:

1. Take high-quality images
2. Import images
3. Prepare images for analysis

The actual processing steps are secondary:

4. Apply algorithm
5. Extract outcome measures
6. Display and interpret results

For any image processing, the first step is taking high quality images:

Qualities of a good image:

- Image is not too dark or too bright. In particular:
  - Pixel values lie within the full bit range (e.g. 0-255 for 8-bit)
  - 0% - 0.5% of pixels are saturated (e.g. very few pixels have a value of 255 exactly).
- Image is in focus: objects are bright and have sharp features against a dark background
- Low noise, e.g. by reducing the gain as much as possible
- Constant illumination/brightness across the image
- Image is saved in a format with no compression; TIF is usually best.

Background on image thresholding, image segmentation, and the watershed algorithm:

Image segmentation and thresholding:

Image segmentation is a broad class of operations where the goal is to look at each pixel in the original image and decide which object or region it belongs to. For example, if you have a fluorescent microscopy image of many cells, you may want to decide if a pixel is part of the background (i.e. not part of a cell), or if it is part of the 1st cell, or the 2nd cell, etc. Thus, the output of any segmentation operation is a new image, called a "label matrix," where each pixel value is an integer number that identifies which region that pixel belongs to (e.g. pixel value=17 if it is part of region #17).

Many people have experience with thresholding an image. Thresholding is related to segmentation, but distinct. In thresholding, you pick a single value as a threshold, e.g. if your pixels have values between 0 and 255 you may pick you threshold to be 100 (somewhere in the range). Then, if a pixel's value is below this threshold, you say it is part of the "background" and set the pixel to be 0. If the pixel value is above this threshold, you say the pixel is part of the image "foreground" and you set the pixel value to be 1. This is a very simplistic way to segment your image into exactly 2 objects: object #0 = background, object #1 = foreground.

- ➢ To implement in MATLAB, use "`t = graythresh(raw_image)`" to get a threshold using Otsu's method and then use "`threshold_image = im2bw(raw_image, t);`" to threshold your image.

Compared to thresholding, segmentation is a higher level operation, where we can have an unlimited number of objects, instead of just two. One straightforward method of segmentation is to start with your threshold image and say that each "connected blob" of foreground pixels is a unique object. In many cases this is a great way to segment objects.

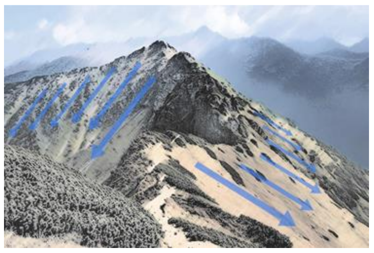
- To implement in MATLAB, apply the function "`label_matrix = bwconncomp(threshold_image);`"

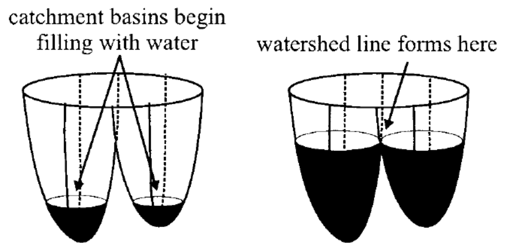
However, this method can be problematic sometimes, especially when objects are touching or overlapping, because they will all be counted as the same object, instead of different objects. The watershed transform is another method of segmentation that is a bit more complex but is particularly good at separating toughing or overlapping objects. This is particularly useful in my research because I analyze images of chondrocytes (cells) in articular cartilage. Usually, these cells are separated from each other, but sometimes two or three are smashed right next to each other. Cells like these are very difficult to tell apart when you are using a simple thresholding operation.

Image segmentation using the watershed transformation

To perform watershed segmentation, we first think of our 2D image as a topographic map that shows hills and valleys. In our map/image, the value of each pixel corresponds to the elevation at that point. Thus, dark areas (low pixel value) are valleys, while bright areas (high pixel value) are hills.

The watershed transform finds catchment basins (or dark areas) in this landscape. For example, think about a mountain ridgeline. When it rains, water that falls just to one side of the ridge will flow down into one lake. However, rain that falls on the other side will flow down into a different lake. Thus the ridgeline at the top of the mountain is also a line that divides these two catchment basins.


http://www.next.cc/journey/language/watershed


catchment basins begin filling with water
watershed line forms here
http://iacl.ece.jhu.edu/~prince/ws/

The watershed transform performs this same analysis on your image. The result is a set of unique catchment basins with ridgelines between them. Each unique catchment basin is a different region (a.k.a. segment). Each pixel belongs either to one region or to the system of ridgelines. Thus, the watershed transform outputs a label matrix where each pixel value is an integer that identifies which region it belongs to: "0" if it is a ridgeline, or "1" if it is part of region #1, or "2" if it is part of region #2, etc. Because it is looking for valleys (dark areas), it is important to use an image which has dark objects on a bright background. In many cases, images show bright objects on a dark background. If this is the case, you need to invert your image before running the analysis.

What to do after segmentation?

After you have segmented your image to identified unique regions (e.g. via watershed segmentation or some other technique), you can easily perform many interesting and useful calculations. For example, you can find the centroid of each cell and use this to calculate the number density of cells across your image. Alternatively, you can find the mean pixel intensity of each region to see if some cells look different than other cells. At this point the world is unlimited! At this stage in your analysis, you can even start to apply data science tools such as clustering and machine learning to extract even more insight into the behavior of your objects.

MATLAB example of segmentation using the watershed transformation

To illustrate watershed segmentation, here I walk through an example case in MATLAB where I use a watershed-based algorithm to find cells in the image "`EthDzoom.tif`." At each stage, I will display the result to a figure. See the MATLAB file "`watershed_example.m`" for this demo.

Note about marker-controlled watershed segmented

In this demo, I actually use "marker-controlled" version of watershed segmentation. What this means is that I force the image to have mountains and valleys where I want them *before* I run the algorithm. In fact, I use thresholding to identify "markers" where I *know* there should be mountains or valleys and then I enforce this topography. In this demo, I enforce 3 things:

1. I want the background to be on big catchment basin (which I will later discard), so I set all the background pixels to be one big flat valley with a low pixel value;

2. I want there to be a big mountain between the background and the foreground, so I set the pixels at the boundary of the background to be very high;

3. I also want my cells to be catchment basins. To help this process, I use a severe threshold to identify pixels that I *know* are part of the foreground. Like the background, I also set these pixels to be valleys. At this step, it is important to make sure that this thresholded image does not have multiple cells in one "blob," otherwise that blob will always look like only 1 object, instead of multiple objects. However, it *is* ok if you miss some objects when identifying this foreground.

You can use the watershed algorithm without this "marker-control" and it may work well – it depends on your image. Similarly, you could use various other methods to pick your "markers" that you enforce. However, the overall message is that you can drastically change the watershed results by varying if and how you enforce any topographical features.

Outline of steps in the watershed segmentation demo:

This is an outline of how I implement the watershed algorithm in the MATLAB file "`watershed_example.m`":

1. Import image into matlab
   a. Use the built-in function "`imread`" to read the image file as a 2D array of 8-bit unsigned integers (i.e. integers from 0-255)
2. Prepare image for analysis
   a. Blur with a Gaussian to remove high-frequency noise.
   b. Apply a median filter and subtract this from the image to remove the background
3. Apply watershed algorithm
   a. Use multiple-Otsu thresholding to identify foreground and background pixels
   b. Invert the background-subtracted image
   c. Enforce known topography:
      i. set background pixels to be low
      ii. set pixels at the boundary of the background to be high
      iii. set foreground pixels to be low
   d. Perform the watershed transformation to identify individual cell regions
4. Use the segmentation to calculate cell density in the horizontal direction
   a. Use "`regionprops`" function to calculate the centroid of each region
   b. Calculate the number of cells and use this to find the cell density (units: number of cells / square pixel)
   c. Bin the x-centroids to determine the cell density in the x-direction and plot the result

Throughout this demo, there are 3 main parameters that you can use to tune the results:

1. `sigma` – [Step 2.a, Line 13] This is the width of the Gaussian used to smooth the raw image and reduce noise (units: pixels). This should be about 10-50% of the size of an object.
2. `med_filt_size` – [Step 2.b, Line 23] This is the size of the median filter used for background subtraction. This should be significantly bigger than one object but also must smaller than the size of your image (units: pixels).
3. `num_bins` – [Step 4.c, Line 74] This is the number of bins in the X-direction to use when computing the histogram of cell positions (used to determine cell density). This should be small enough so that there are at least 5 counts in each bin.

Content of “watershed_example.m”

```matlab
%% 1. Read image data

% Read image as an array
path_to_image = 'C:\path\to\your\image\EthDzoom.tif';
raw_image = imread(path_to_image);

figure('name', 'raw image')
imshow(raw_image)

%% 2. Prepare image for watershed algorithm

% Gaussian blur to remove noise
sigma = 1;                                         % *** Parameter ***
hsize = ceil(sigma*6);
if ~mod(hsize,2), hsize = hsize+1; end % size must be odd
gaussian_filter = fspecial('gaussian', hsize, sigma);
blur_image = imfilter(raw_image, gaussian_filter);

figure('name', 'gaussian blur')
imshow(blur_image)

% Median filter subtraction to remove background
med_filt_size = 10;                                 % *** Parameter ***
median_bg = medfilt2(blur_image, med_filt_size*[1 1]);
bg_removed_image = blur_image - median_bg;
imshow(bg_removed_image)

figure('name', 'foreground/background identification')
imshow(bg_removed_image)

%% 3. Apply watershed algorithm

% Identify image foreground and background using multi-Otsu thresholding
thresh = multithresh(bg_removed_image, 2);
bg_image = bg_removed_image < thresh(1);
fg_image = bg_removed_image > thresh(2);

figure('name', 'foreground/background identification')
imshowpair(raw_image, bwperim(bg_image) | bwperim(fg_image))

% Prepare image for watershed transformation:
% 1. Inverting the image, so objects are dark (i.e. low/minima)
% 2. Enforce desired watershed topography, by setting:
%    a. foreground and background pixels to be low
%    b. pixels at the boundary of the background to be high
image_for_watershed = -mat2gray(bg_removed_image);
image_for_watershed(bg_image) = -1;
image_for_watershed(fg_image) = -1;
image_for_watershed(bwperim(bg_image)) = 1;

figure('name', 'image to input into watershed')
imshow(image_for_watershed, [])

% Perform watershed transformation to identify local basins - these are cells
watershed_regions = watershed(image_for_watershed);

figure('name', 'regions identified by watershed')
imshowpair(raw_image, watershed_regions==0)

%% 4. Extract outcome measures: average cell area and cell density in the x direction

% Use regionprops to extract the centroid of each region
% Ignore the 1st region, because this is just the background
stats = regionprops(watershed_regions, {'Centroid'});
all_centroids = cat(1, stats.Centroid);
cell_centroids = all_centroids(2:end, :);

% Compute the average cell number density, i.e. Number of cells per area
num_cells = size(cell_centroids, 1);                % Total number of cells
avg_cell_density = num_cells / numel(raw_image);    % Number of cells per square pixel

% Bin the x-component of the centroids to determine the cell number
% density
num_bins = 100;                                     % *** Parameter ***
num_x_pixels = size(raw_image, 2);
bin_edges = linspace(0, num_x_pixels, 100)';
centroids_x = cell_centroids(:,1);
bin_counts = histc(centroids_x, bin_edges);

figure('name', 'Cell density in the x-direction')
bar(bin_edges, bin_counts, 'histc');
title('Histogram of cell X-positions')
xlim([min(bin_edges) max(bin_edges)])
```

Image to be analyzed, “EthDzoom.tif,” and the corresponding segmentation output: