An informal image analysis course

Prof. Raghuveer Parthasarathy

The University of Oregon
Email: raghu@uoregon.edu

Last modified: July 15, 2020

Background and Motivations

In 2014, 2015, and 2016 I taught an informal ("off the books") course on computational image analysis. The goal was to help people (1) learn about methods for extracting information from images, and (2) improve their skills in writing code to implement these methods. The things we image, the ways we image them, and the assessments we want to make based on images are all extremely varied, and the ability to develop custom software tailored to specific needs is invaluable. In addition, familiarity with common image processing methods is useful for planning and discussing experiments, in part to better assess what tasks are easy or difficult. Rather remarkably, there are no formal courses on image analysis offered at the University of Oregon as far as I know, providing additional motivation for this course. The subjects we cover also intersect important topics in statistics and data analysis, such as maximum likelihood estimation, that are not typically taught to biology or physics students.

The audience for the course was graduate students, postdocs, and advanced undergraduates in physics and biology. As it was an informal course (taught outside my usual teaching load), and the students all had considerable intrinsic motivation, we met just once a week and did not have any assessments such as quizzes or exams. During the weekly meetings we discussed problems with the previous week's topic and assignments, shared results, asked questions, and introduced the next module. The sessions were quite enjoyable. Approximately 10 people participated each time – the number would likely be higher if I more actively advertised the course.

This document describes the topics, readings, and assignments for the course. Several sections need improvement, both so that this will be a more useful stand-alone resource, and so that I can improve the course next time!

Syllabus

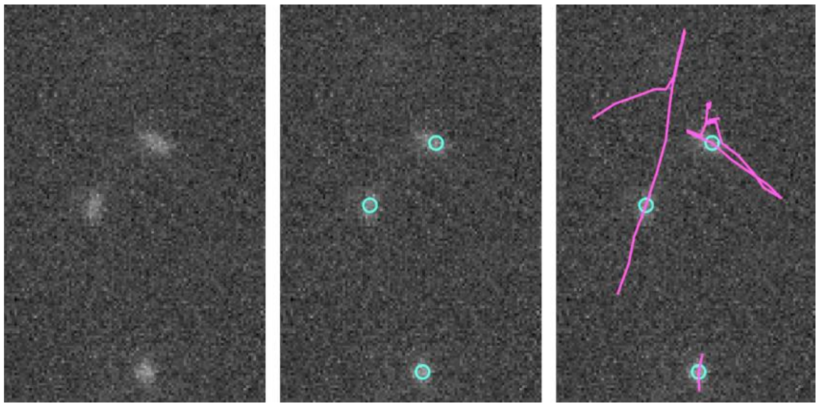


Figure 1. Given an image of bacteria (left), how do we accurately determine their positions (center), and their trajectories across a series of images (right)? Understanding algorithms for quantitative image analysis lets us perform these and countless other tasks.

Course Goals

Students will learn about a variety of algorithms for computational image analysis and will practice writing code to implement them. The methods we'll explore are very broadly applicable in many areas of science and technology, and we'll see insights they enable in biophysics, soft condensed matter physics, microbiology, and perhaps other fields of interest to students. We will also encounter important topics in statistics and data analysis, such as maximum likelihood estimation, gaining an understanding of their meaning and utility.

Course structure

Since this is a reading / self-study course, and since I've got lots of other things to do, my own involvement will be rather minimal. We will all meet roughly once a week to introduce topics and discuss the work of the past week, but students will be expected to read on their own, to initiate discussions with other students, and to develop implementations of algorithms quite independently.

We'll use a Google Drive folder to share articles, images, etc. URL: https://drive.google.com/drive/folders/11bSZ5DeHzX1u7gn8TTCrNLyCC3iQUcHm?usp=sharing

The course will move quickly, spanning at least one new method or concept per week.

Prerequisites and languages

Students should have a good grasp of some high-level programming language and should be comfortable with manipulating arrays and writing functions.

All of my examples will use **MATLAB**[1], which has useful features like logical indexing and an excellent toolbox of basic image processing functions. See this footnote[2] on MATLAB

[1] I've put a very short MATLAB tutorial I wrote ("MATLAB tutorial RP.pdf") in the Google Drive folder. Be able to do the exercises there. For a longer introduction to MATLAB, see "Essentials of MATLAB Programming by Stephen J. Chapman, which is on the lab bookshelf, or this guide by Philip Nelson: https://github.com/NelsonUpenn/PMLS-MATLAB-Guide. It's a 70 page PDF; you'll have to download the entire ZIP of the repository. I haven't read it but I expect that it's very good.

characteristics with which one should be familiar. Students are welcome to use other languages, especially **Python**, though I may not be able to help with problems as well as with MATLAB.

On coding: always try to write clear, readable code! Imagine that you're looking back on what you wrote years later: Would it make sense? Could you build on it? I strongly recommend the "PLOS Biol" cited in this footnote[3].

Sources

In addition to my own thoughts and past efforts, the syllabus draws heavily from:

- Stanford University, EE 368 / CS 232 (Bernd Girod): http://www.stanford.edu/class/ee368/index.html; Notes and handouts at http://www.stanford.edu/class/ee368/handouts.html
- Gonzalez, Woods, & Eddins – *Digital Image Processing Using MATLAB* (There's a copy in my lab.)

Topics

[2] If you're not already, you should become familiar with how to make MATLAB code fast – using logical indexing, avoiding for-loops, preallocating memory, etc. See e.g. http://www.csc.kth.se/utbildning/kth/kurser/DN2255/ndiff13/matopt.pdf , http://web.cecs.pdx.edu/~gerry/MATLAB/programming/performance.html

[3] G. Wilson, ..., P. Wilson, Best Practices for Scientific Computing. *PLOS Biology*. **12**, e1001745 (2014). https://journals.plos.org/plosbiology/article?id=10.1371/journal.pbio.1001745

Module 1 Introduction / fundamentals / thresholding

In which we consider basic properties of digital images and apply thresholds to enhance contrast.

COMMENTS: This was originally combined with Module 2, which together was too massive. Many students will already be familiar with the contents of this module, and so could skip it.

Readings

- Gonzalez, Woods, and Eddins, Chapter 2 ("Fundamentals"). Most of Chp. 2 is available at: http://www.imageprocessingplace.com/DIPUM-2E/dipum2e_sample_book_material.htm
- [Optional] Video: "Microscopy: Introduction to Digital Images" by Kurt Thorn, https://www.youtube.com/watch?v=SNl3kRBgW10. Especially the first 10 minutes.
- Read about Otsu's method for segmentation (Wikipedia, or Section 11.3 of Gonzales et al. – I'll scan this.) You needn't remember the details, just what the general idea is.

Assignment

- **Preliminaries.** (i) Read the MATLAB tutorial, on logical indexing and for-loops. (ii) Image arrays are typically unsigned integers; to perform some mathematical operations you may have to convert them to double precision numbers (e.g. `A = double(A)`); (iii) Read the documentation for MATLAB's `imshow` function. The syntax `imshow(A, [])` is often useful. (If you're using a different language, read and figure out the equivalent things.)
- **1.1 A puzzle.** Run the following (MATLAB). Note that "kiran1.tif" is in the Drive folder.

```
Image_A = imread('kiran1.tif');
figure; imshow(Image_A, []);
Image_B = Image_A*6.2;
Image_C = Image_B/6.2;
figure; imshow(Image_C, []);
```

  ??!! Why is `Image_C` not the same as `Image_A`, since multiplying and dividing by 6.2 should give us the same thing we started with? Try this instead:

```
Image_B = double(Image_A)*6.2;
Image_C = Image_B/6.2;
figure; imshow(Image_C, []);
```

  Why does this work?
- **1.2 Thresholding.** Practice thresholding, both global and local, as described in the Gonzalez *et al* book. Find some images; try things. Be ready to share an example.
- **1.3 More Thresholding.** Use Otsu's method (`graythresh` in MATLAB) to determine the threshold for the images "robert-mapplethrope-calla-lily-1984.png" and "MakeUp_RichardPrince_1983_gray.png" (in \Drive). Comment on the results. Look at the histogram of pixel values for each image. Also threshold an inset to the Calla Lily photo: "robert-mapplethrope-calla-lily-1984_CROP.png" – why does this look different than the thresholded whole image?

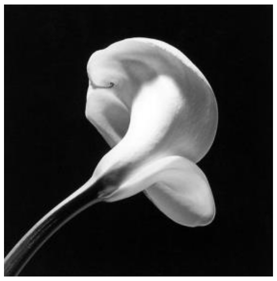


Figure 2. *Calla Lily* -- Robert Mapplethrope, 1984

Module 2 Spatial filtering

In which we examine various local filters.

COMMENTS: As noted above, this was originally combined with Module 1.

Readings

- Gonzalez, Woods, and Eddins, parts of Chp. 3: Read most of 3.3-3.5. Section 3.3 has a lot of tips about plotting in MATLAB. Skip 3.3.3-3.3.4. Sections **3.4** and **3.5** are important.
- Note that parts of this chapter are available at: http://www.imageprocessingplace.com/DIPUM-2E/dipum2e_sample_book_material.htm

Assignment

- **2.1 Convolution and filtering.** Make a square array of ones (`ones` in MATLAB) and use `conv2` to perform a local smoothing of an image by convolving it with the "ones" array. Do the same using `imfilter` and show that it gives the same result. Verify by hand (i.e. looking at intensity values of pixels in your image) that a pixel in the output image is the sum of pixels in its neighborhood.
- **2.2 Gaussian filter.** Create a 2D gaussian filter, i.e. a 2D array in which the value is a Gaussian function of the distance to the center of the array. In MATLAB, use `fspecial`; see the documentation for its parameters. Filter some image using this Gaussian filter.
- **2.3 Median filter.** Use `medfilt2` to perform a local median filtering of your image. Compare Gaussian and median filtering, looking especially at sharp edges in your image.
- **2.4 Filtering and thresholding.** Apply a Gaussian or median filter to Module 1's "MakeUp_RichardPrince_1983_gray.png" photo, and then threshold the image.
- **2.5 High-pass filter.** Download "k_icebird_circles_28Mar2014.png" (shown below) from Dropbox. How can we remove "smudges" and retain lines? Smudges are (roughly) unchanged by Gaussian smoothing; lines are blurred away. Therefore *subtracting a smoothed image from the original image* acts as a "high pass filter," retaining sharp features like lines. **Create a sharper image** like that of "k_icebird_lines_30Mar2014.png" (or better-looking, since I didn't spend much time on this) by performing these transformations. You may have to invert the image – for an 8-bit image "`A`", inversion is `255-A`, since this takes black to white and vice versa. You can also try median, instead of Gaussian, filtering. There are many steps involved in this exercise.

Other things to discuss

- Color images – why we typically don't want them, and how to deal with RGB images.

If we had more time...

Readings:

- Gonzalez, Woods, and Eddins, Chp. 3-4; Girod EE368, Handout #8
- Merz et al. "HiLo" papers, e.g. Merz and Kim, *Journal of Biomedical Optics* **15**, 016027 (2010)

Assignment:

- "HiLo" structured illumination: implement and assess...

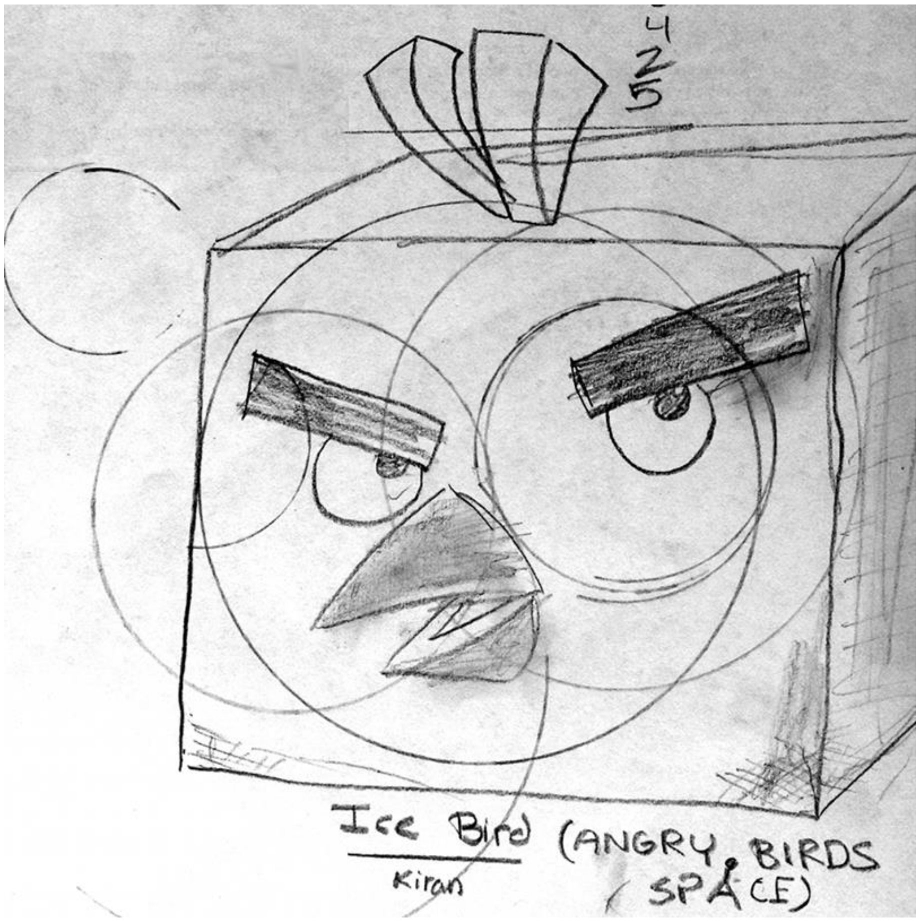


Figure 3. *Ice Bird*, by Kiran Parthasarathy (2014).

Module 3 The point-spread function, simulated images, noise

In what way is an image different from the reality it represents? How is noise manifested, and how does that limit what we can learn from an image?

COMMENTS: This is a challenging module. To make it self-contained, I should include a simple introduction to what the point spread function is, and a description of how the point-source simulation (Ex. 3.2) works. (Of course, I can describe all this in the preceding class.) Note added: I've done this, and also made a video: https://www.youtube.com/watch?v=wK5gqxLk30w

Readings

Just read the parts of these articles that are about simulated image construction:

- Sections 1-3 of Parthasarathy and Small, "Superresolution Localization Methods," *Annu. Rev. Phys. Chem.* 2013. **65:**107-125. (*http://www.annualreviews.org/doi/abs/10.1146/annurev-physchem-040513-103735*)
- R. J. Ober, S. Ram, E. S. Ward, "Localization Accuracy in Single-Molecule Microscopy," *Biophys. J.* **86**, 1185–1200 (2004). *https://www.cell.com/biophysj/fulltext/S0006-3495(04)74193-4*
- The first few sections of Materials and Methods, page 2 through "noise model" on page 3, of: M. K. Cheezum, W. F. Walker, W. H. Guilford, "Quantitative comparison of algorithms for tracking single fluorescent particles," *Biophys J* **81**, 2378-2388 (2001). https://www.cell.com/biophysj/fulltext/S0006-3495(01)75884-5
- [Optional] Regarding super-resolution and the challenge of finding objects, you might like this video, *http://www.ibiology.org/ibioseminars/cell-biology/xiaowei-zhuang-part-1.html*, which gets at the methods described in the first section or two of the Parthasarathy and Small paper.
- [Optional, possibly of interest to those who want to know more about optics.] A few pages on point-spread-functions: Chapter 3, through "circular lenses." M. Gu, *Advanced Optical Imaging Theory*, Springer, 1999. A PDF is posted: "Gu Chapter 3 through 3p2p1 PointSpreadFunction.pdf". **Note** that λ here is not the free-space wavelength, but λ_{free}/n.

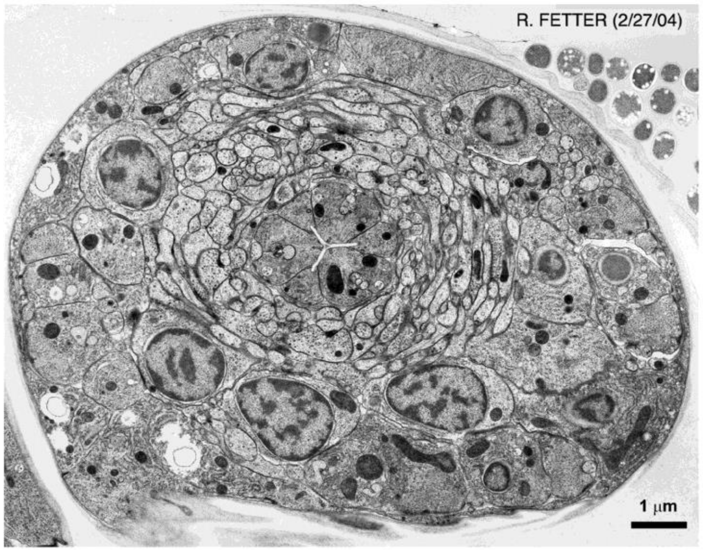


Figure 4. A TEM image of a slice through a C. elegans embryo. *Source:* Rick Fetter and Cori Bargmann.

Assignment

- **3.1 A worse worm image.** Download "fetter_Celegans_cellfig10.jpg" from Dropbox (\...\Week 2 -- Simulated images and noise), shown above, a transmission electron micrograph of a slice of a *C. elegans* embryo, from Rick Fetter and Cori Bargmann (source: http://www.wormbook.org/chapters/www_intromethodscellbiology/intromethodscellbiology.html). Note the scale bar. Manipulate this image create an image that shows what the slice would look like if imaged with visible light (e.g. wavelength 530 nm) rather than electrons. You can write your own function to calculate a point spread function (recommended) or use `psf2d.m`. Don't worry about simulating noise or pixelation.
- **3.2 Simulated point sources.** Use `modelimage.m` to create several simulated images of single point sources (e.g. single molecules) for several different signal-to-noise ratios (SNr), i.e. some number of detected photons[4]. Understand how `modelimage.m` works – if there were more time, I'd ask you to write something like it yourself. For simplicity, place the centers of all the simulated images at the origin (0,0). Note that the simulated image *is* the probability distribution p described e.g. in Parthasarathy and Small Equations 3-4.
- **3.3 Simulated bacteria.** Write a version of `modelimage.m` that simulates images of rod-like bacteria, rather than images of point sources. Have the length, width, and orientation of a bacterial cell be input parameters; typical values are 1x3 microns. Alternatively, simulate some other shape that is relevant to your research, such as "Y"-like junctions between lines that one might find in images of branching neurons. To help you, I have provided `modelimage_ring.m`, which simulates images of rings – you can read and modify this.
- **3.4 The CRLB [Optional].** Use Eqns 1-4 to calculate the Cramer-Rao Lower Bound (CRLB) for your simulated images, numerically evaluating the second derivative of p with respect to x. For point sources, by the way, the CRLB can be calculated analytically, but it can also be determined by brute-force as you're doing here. If you're having trouble, the "answer" is in `CRLB_from_modelimage.m`.
- **3.5 Limits on accuracy [Optional].** How would you determine the fundamental limit on accuracy for determination of (i) the orientation angle of a rod-like bacterium, and (ii) the location of the branching points of an axon. You don't have to write a program; just comment on how (or whether) this is possible. In each case, assume one has a fluorescent bacterium or neuron (i.e. a bright image on a dark background), and that one can isolate a single bacterium or branch point per image.

Notes for students

- For convolving an image with a (2D) PSF, you should use "conv2" or "imfilter" in MATLAB. MATLAB has several convolution functions, "conv" for 1D arrays, "conv2" for 2D arrays, "convn" for N-dimensional arrays.
- It's useful to use "surf" to visualize things like the PSF, for example:

```
psf = psf2d(100,0.01, 0.9, 0.53); % A PSF with a scale of 0.01
microns/px, NA = 0.9, wavelength 0.53 microns
figure; surf(psf); shading interp
```

[4] As constructed, `modelimage.m` returns the number of photons detected at each pixel as its output. The peak brightness is SNr^2, and the sum of all the pixel values is the total number of photons.

Module 4 Sub-pixel localization

How can we determine the location of a point source (e.g. a single molecule, or a colloidal particle)? In addition to being the key issue underlying super-resolution microscopy, this is applicable to many "tracking" problems.

COMMENTS TO MYSELF: Exercise 4.1 could be assigned earlier.

Readings

- Parthasarathy and Small, "Superresolution Localization Methods," *Annu. Rev. Phys. Chem.* 2014. **65:**107–125. (In the Readings folder.) See also some of the papers it cites.
- [Optional] R. Parthasarathy, Rapid, accurate particle tracking by calculation of radial symmetry centers. *Nat. Methods.* **9**, 724–726 (2012).

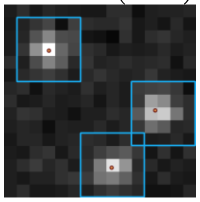
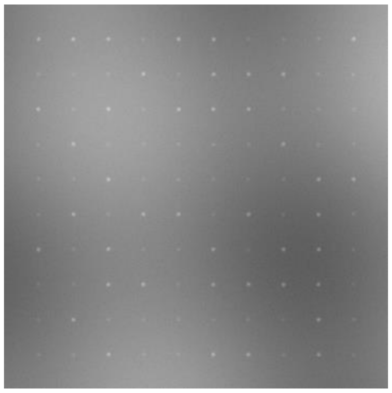


Figure 5. From Parthasarathy and Small (2014), Fig. 2



Assignment

- **4.1 Quantized aggregates.** Download "emitters2_33px_dim_hard.png," a simulated image of 100 point-emitters arranged on a grid. Each emitter is located at a multiple of (33, 33) pixels. Imagine that these are images of single-molecules, a GFP-tagged protein, for example. You want to know: are these proteins monomers, dimers, trimers, ... or a combination of these? (In other words, is the brightness of the dots quantized, and if so, how many quanta are there per dot?) Except for the placement on a grid, this is a very "real-world" problem – in fact, it was inspired by a graduate student talk I attended a few weeks ago! **Notes:** (1) You're only interested in brightness for now, so this problem doesn't require any of the localization concepts of this module – in fact, I could have assigned it last week. (2) There are, of course, background intensities of various sorts which you'll have to figure out how to deal with. Take the positions (33,33 px) as a given, though. (3) The same image without any noise, except shot noise, is at "emitters2_33px_dim_hard.png," and images with brighter emitters are at "emitters1_33px_bright_{easy, hard}.png" – try your analysis on these, first.

- **4.2 Centroids.** Write your own function to use simple centroid finding, i.e. the "center of mass" of the image, to estimate the location of a point source in an image.
- **4.3 Assessing centroids.** Apply your centroid-finding function to a large number (around 1000) of simulated single-particle images (e.g. from `modelimage.m`), keeping (for now) the true object location at the center of the simulated images. Calculate the localization error, i.e. the root-mean-square deviation of the true center from the calculated center. Use a few different values of SNr of the simulated images, and see how the rms error depends on SNr. NOTE (added later): Be sure to have a sizeable background intensity, at least 10.
- **4.4 ... continuing...** For fixed SNr, simulate images with a few different center positions: exactly in the middle of the central pixel, 0.25 pixels in x away from the center, and on a diagonal. Does this affect the accuracy of the centroid algorithm? In what way? (You should discover why the centroid method is a dangerous one!)
- **4.5 Gaussian fitting.** Look at RP's function to fit a Gaussian to a 2D image using maximum likelihood estimation (`gaussfit2DMLE.m`). Apply this to the same images you analyzed in the preceding exercise. Compare its accuracy and speed of execution (use `tic` and `toc`) with centroid-finding. (Readings in the next module will provide more explanation of how this fitting works.)
- **4.6 Radial symmetry based localization [Optional, but recommended].** Look at RP's function to estimate the center of a particle using a symmetry-based algorithm (`radialcenter.m`); see Parthasarathy 2012 for details. Apply this to the same images you analyzed in the preceding exercises, and again compare accuracy and speed of execution.
- **4.7 Quantized aggregate positions [Optional].** Return to the images of Exercise 4.1. The positions aren't exactly at the grid centers – I've added jitter of at most ± 0.25 pixels in each direction. Find the particle locations and compare to the true values tabulated in the csv file ("emitters2_positions.csv"). The values are given in nanometers, and the scale is 100 nm/px. [**Note:** the positions are ordered for the emitters arranged left-to-right, then down. *(I should change this next time. Also, use smaller dhr, to avoid slight offset error. – RP.)*] Use whatever algorithms you like, but use more than 1 and compare.

Module 5 Maximum Likelihood Estimation

In which we learn and apply a bit of statistics.

COMMENTS TO MYSELF: This was added by popular demand! We also discussed the Cramer-Rao Bound in this module and the one before it. All the exercises are optional and somewhat redundant with each other; I suggest doing at least one of 5.2-5.4.

Reading

- From Chapter 7 of Steven Kay, "Fundamentals of Statistical Signal Processing"
- Raghu's, "MLE for fitting a PSF to a signal with Poisson noise," which describes how this works. In Dropbox.

Assignment

- **5.1 Gaussian Noise [Optional].** Note that `gaussfit2DMLE.m` considers a Poisson noise model – in other words, the likelihood function is a Poisson distribution. How would this change if we had Gaussian-distributed noise (i.e. the intensity at each pixel is its "true" value ± a Gaussian random number)?
- **5.2 Airy fit [Optional].** Look at RP's function to fit a Gaussian to a 2D image using maximum likelihood estimation (`gaussfit2DMLE.m`). Make a version that instead fits an Airy function (the actual theoretical form of the point spread function in the image plane).
- **5.3 Bacteria fit [Optional].** Take the simulated bacteria image function that someone wrote last week and write a function that fits an asymmetric 2D Gaussian function to it, using maximum likelihood estimation.
- **5.4 Two particle fit [Optional].** Write a function to find the positions of **two** point objects in an image by MLE fitting of a sum of two Gaussians. Use RP's "`modelimage_Npoints.m`" to create simulated images of two particles, and see `make_2_particles.m` for an example of how to call it. (This also uses the 2D PSF function of last week.) Consider the usual Poisson-distributed noise. Your function should output x and y coordinates for each particle, which you can compare to the true positions.

Module 6 Morphological image processing

In which we familiarize ourselves with an immensely useful "toolkit" of image processing operations.

Readings

- Gonzalez, Woods, and Eddins, Chp. 10
- MATLAB's help documentation on Morphological Filtering
- Girod, Handout #7 (less useful) "7-Morphological_16x9.pdf"

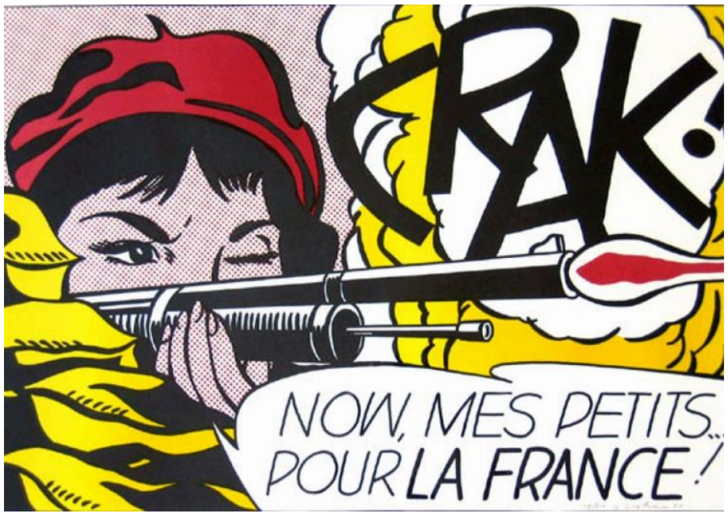


Figure 6. "Crak ! Now, Mes Petits… Pour La France !" by Roy Lichtenstein

Assignment:

- **6.1 Local maxima.** Write a function that finds the positions of local intensity maxima in images using dilation. (As discussed in class, pixels whose value equals the dilated image value are local maxima.) The output should be a list of positions, and the intensities of those local maxima.
- **6.2 Smooth leopard spots.** Take either the color or gray scale leopard fur photos "leopard_fur.jpg" or "leopard_fur_gray.jpg" (which I downloaded from http://www.mindblowingpicture.com, original source unknown), and modify them so that the dark and light regions aren't so ragged due to individual hairs, but rather are smoother and more "blocky." (Imagine, for example, that your goal is to quantify the area of dark and light regions on a leopard, or to construct a library of characteristic rosette shapes, for which the "noise" of individual hairs is irrelevant.) Compare the effects of image opening with closure of the inverse image (255-image). Also try using "morphological reconstruction" (MATLAB's `imreconstruct` function.) The help documentation isn't as clear as it should be about this; explore.
- **6.3 Removing background texture.** Take Roy Lichtenstein's "*Crak ! Now, Mes Petits… Pour La France!*" (1963) (roy-lichtenstein-pop-prints-crak.jpg, downloaded from http://chloejohnston.com/paris-expo-retrospective-on-roy-lichtenstein/), and modify it as

best you can so that (i) all the red dots are removed, leaving only white in these regions, and (ii) all the dots are replaced with "X"s that connect to other "X"s. For both of these, you'll use openings or closings (which are combinations of dilations and erosions). Think about what structuring elements would be appropriate. You can process the color image, acting on all channels (Red, Green, Blue) together. You might find it interesting to look the different color channels. (Note that R, G, B are the three layers of the 3D color image array.)

- **6.4 Plant cells revisited.** We briefly looked at a fluorescence image of a piece of a plant root ("Arabidosis_2a_20X_gfp_tubulin_crop.png"), using morphological image processing to identify nuclei. Do a better job of this, perhaps with better morphological operations, or combining these with filtering operations.
- **6.5 Protein aggregates revisited [Optional].** Use your function for finding local maxima to identify regions of interest in the simulated "protein aggregate" images of Module 4. Combining this with Exercise 4.7 ("Quantized aggregate positions"), you should have a complete set of functions that takes an image, finds objects, and precisely locates the emitters – a very useful package!

Module 7 Image segmentation Part 1

One of the most important, and hardest, problems: what are the "objects" in an image, and how can we identify them – not just their centers, but their size and shape?

Readings

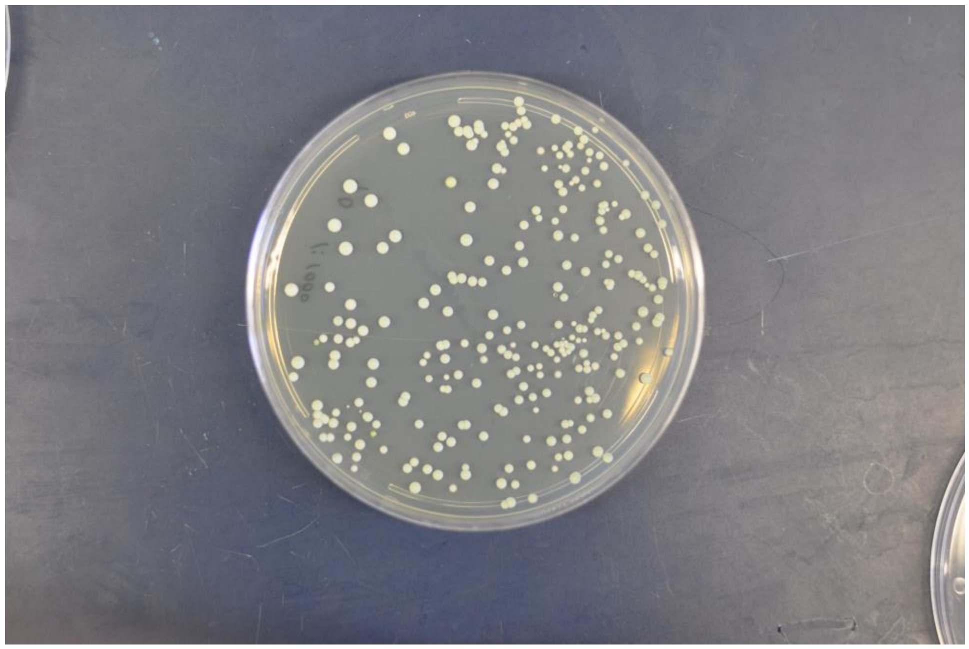
- Gonzalez, Woods, and Eddins, Chp. 11 (through 11.2; note that we previously read 11.3, on Thresholding). The book discusses lots of different edge detection algorithms; don't pay attention to the details, or memorize these – get the general idea, and use this as a reference.
- On Hough transformation of circles: Pedersen Circular Hough Transform 2007.pdf, and http://basic-eng.blogspot.com/2006/02/hough-transform-for-circle-detection.html

Assignment

- **7.1 Edges.** Find some images, and practice using the edge detection functions described in Gonzalez, Woods, and Eddins.
- **7.2 Cell edges.** See the image of cultured epithelial cells "rptec-sa7k-cells.jpg" (from https://www.sigmaaldrich.com/technical-documents/articles/biology/rptec-drug-models.html). Apply various edge detection methods and see how well you can highlight (threshold) the edges of the cells. Crop part of the image, if you want. You may find it useful to apply the "hy" etc. gradient arrays in my "sobelN.m," which makes Sobel filters of larger sizes than MATLAB's defaults; the header comments give an example of how to use this function to calculate gradient magnitudes. Don't spend much time on this.
- **7.3 Hough Transformation for Circles.** The Gonzalez *et al.* book describes a Hough transformation for detecting lines. Another clever and common use of a (different) Hough transformation is for detecting circles and locating their centers. Note our sketch of how this worked at our July 1 Zoom session, and see the reading noted above.
  (i) Either write your own function that implements a Hough transformation, or use my "`houghring_rp.m`" function. Your function should take an image and a radius value as the inputs, and output the circle's center *and* the "parameter-space" image ("HM2" in my function). Apply Hough transformations to whatever images you want, or to the "simRing_{1,2,3}.png" images that I have provided, for which $R = 25$ pixels. What does the parameter-space image look like when you input the values of R other than the "correct" value of 25? See `Hough_transformation_snippet_RP5July2020.m` for a snippet of code illustrating how to use `houghring_rp.m`
  (ii) As discussed, the Hough transform is slow! (Why?) Can you process your image to make it faster? Try it, or look at the code in my "`houghring_rp.m`" function.
  (iii) I've put three simulated rings with $R = 25$ pixels in the image "simRing_threeRings.png" . Using a Hough transformation, find the ring centers. To the nearest pixel, you should find x = [70 190 301]; y = [280 215 140]. More precisely, the "true" values are x = [0.2, -0.3, 0.4]; y = [-0.1, 0.1, -0.3] pixels, relative to the above center pixels.
- **7.4 Yeast colonies, part (i).** [Optional; useful if you want to get a head start on the next module.] See the image of yeast colonies on a petri dish, "dsc_0357.jpg" (from http://sciencebrewer.com/tag/brett-agar/). The *overall* goal (continuing to next week) will be to write a function that, with a few user-input parameters returns the location of each colony, its size (pixels), and eccentricity (a flag for potential errors in detection). *To start:*

Apply whatever filtering seems appropriate, and threshold to make a binary image that highlights the colonies. (Colonies = 1, background = 0.)

- **7.4 Yeast colonies, part (ii).** In your filtered image, find and display these maxima using MATLAB's "`imregionalmax`" function or by using dilation to identify maxima, as in the last module. (Use "find" to get the coordinates of the local maxima and plot a red 'x' at each location.) You should find far too many maxima; try combining with the thresholded image, to only keep the "bright" maxima.
- **7.4 Yeast colonies, part (iii).** Apply morphological operations to the original or the thresholded image that help separate colonies that are touching each other. (Don't expect great improvements.)
- **7.4 Yeast colonies, part (iv).** Use `regionprops` to get statistics on the various regions – i.e. connected above-threshold pixels. Calculate each region's area and eccentricity, and whatever else you want. Make a plot of eccentricity vs. log(Area). (We'll see `regionprops` more next week. There's a Python version in `skimage.measure`.)



Module 8 Image segmentation Part 2 – Watershed Segmentation

Continuing the topic of image segmentation, we'll move on to "region based" methods, especially watershed segmentation.

Readings

- Gonzalez, Woods, and Eddins, Chp. 11 (11.4-11.5)
- MATLAB's documentation has a good description of marker-controlled segmentation (also available at https://www.mathworks.com/help/images/marker-controlled-watershed-segmentation.html). The specific details aren't important, but the general ideas of using morphologically transformed regions as foreground markers and distance transforms as background markers are useful.

Assignment

- **8.1 Yeast colonies.** Do Exercises 7.4 (i) and (ii) if you haven't already. You'll filter and threshold an image of yeast colonies on a petri dish, "dsc_0357.jpg," to get a binary image that provides a crude segmentation of the colonies, and to find local maxima that will serve as seeds for watershed segmentation in a later exercise. It may take some experimentation to come up with good filters; keep in mind both high- and low- pass filtering. (Recall what's useful about each.) For all of these exercises, use a grayscale version of the image, averaging over color channels, or just pick one channel to analyze.
- **8.2 Watershed without markers.** Calculate and display the gradient magnitude of the yeast colony image. Apply watershed segmentation and visualize the results. It should look terrible. To make sure you can move on to the interesting part, here some code that does all this (MATLAB):

```
% Gradient magnitude; "im" is the image
im = double(im);
im = mean(im,3); % average over color channels
hy = sobelN(5); % 5 px Sobel filter
hx = hy';
im_gx = imfilter(im, hx, 'replicate');
im_gy = imfilter(im, hy, 'replicate');
im_gradmag = sqrt(im_gx.^2 + im_gx.^2);
figure('name', 'Gradient magnitude'); imshow(im_gradmag, []);

% Watershed segmentation
watershed_im = watershed(im_gradmag);
% This makes each object a different RGB color, for visualization
watershed_rgb = label2rgb(watershed_im,'jet',[.5 .5 .5]);
figure('name', 'Watershed'); imshow(watershed_rgb,[])
```

- **8.3 Watershed with markers.** Using the local maxima as foreground markers (i.e. a binary image that's "1" at the local maxima), whatever seems useful as background markers, and the gradient magnitude as the "topography," use MATLAB's watershed segmentation function to segment the image. See if colonies that touch each other can be recognized as separate objects! Try different markers, such as combining the local maxima with the thresholded image, dilating/eroding, etc. Just assess the quality of the segmentation by eye.

- **8.4 Snakes!** Try applying active contour segmentation to your (filtered) image (function: "`activecontour`" in MATLAB). Note that it requires two arguments, the image and the initial guess for the contours. This initial guess should be something that is clearly smaller than or larger than the true objects – for example, an eroded or dilated binary thresholded image. You can play around with parameters, but don't spend much time on this; you'll probably find that it's slow!
- **8.5 Region properties.** As in 7.4 part (iv), use `regionprops` to get statistics on the various regions you've found from segmentation. Calculate each region's area and eccentricity, and whatever else you want. Make a plot of eccentricity vs. log(Area). (There's a Python version in `skimage.measure`.) *Optional:* Get rid of objects whose properties imply that they're not colonies (e.g. high eccentricity).
- **8.6 Other images.** How well does what you've done work on other images? Try some images of bacterial colonies on plates (from my lab) in \Images\Bacteria on plates\ . Note that all these images are quite easy compared to a lot of segmentation tasks! Try also something much more difficult: the image of zebrafish bone forming cells (Chuck Kimmel's lab) that we saw in class: "Jamie_Zfish_55hpf1_w1YokoRFP_im30_slice24.png". (Or some other image you find.) Imagine writing something to accurately segment hundreds of these...
- **8.7 A more general function.** [*Optional*] Combining all of the above things, write a function that has minimal user input (e.g. specifying the circle containing the dish, specifying a max. and min. colony size, ...), that takes an image of plated colonies as input, that performs segmentation and rejects objects that are too large/small/eccentric, and outputs an image of the found colonies together with lists of colony positions and characteristics. Congratulations: you've automated the tedious task of colony counting!

Module 9 Machine learning methods (an introduction)

*Can we get the computer to do our thinking for us? How? A *very limited* introduction.*

Readings

- Hsu et al. "A Practical Guide to Support Vector Classification" – https://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf
- Some comments on learning about machine learning, with links to Caltech's and other courses (https://eighteenthelephant.com/2015/10/23/learning-about-machine-learning-part-i/, and part ii)
- [optional] On neural networks – "A high-bias, low-variance introduction to Machine Learning for physicists" by Mehta et al. (https://arxiv.org/abs/1803.08823)

Assignment:

- **9.1 Yeast colonies and Support Vector Machines: Training data.** Consider the early part of your Module 8 exercises, in which you identified regions using filtering and thresholding, and determined properties of those regions (area, eccentricity, etc.). Here, we'll use a Support Vector Machine to classify these regions as "colonies" / "not colonies," or "single colonies" / "not single colonies." First, let's make the training data. Write a program that allows you to click on a point in the image, that finds the object closest to where you've clicked, and that adds that object to a list of "colonies;" and the same for a list of "not colonies." Make lists of about 10 objects each.
- **9.2 Support Vector Machines.** There are lots of toolboxes for SVMs. I've often used libSVM: https://www.csie.ntu.edu.tw/~cjlin/libsvm/, which has libraries for MATLAB, Python, etc. MATLAB has its own SVM functions which (at least a few years ago) were very slow, but you might like to try these. Look into the syntax to use, to construct arrays of features and classifiers for training data.
- **9.3 Yeast colonies and Support Vector Machines: Classifying!** Support vectors in hand, have your program classify all the remaining objects in your image. Plot green and red "X"s on colonies and not-colonies, and just assess "by eye" how good it seems. Also, apply the SVM to a new image, e.g. the bacterial colonies in Exercise 8.6.

Module X Particle Linkage

How can we link together images of objects across frames to form accurate trajectories?

COMMENTS TO MYSELF: We've always skipped this. It's not really image analysis *per se*, and in practice I've never moved beyond simple methods. Could make a nice project for someone.

Readings

- N. Chenouard *et al.*, Objective comparison of particle tracking methods, *Nat. Methods* **11**, 281–289 (2014) [*Chenouard meijering comparison tracking methods 2014.pdf*]
- "Inference in particle tracking experiments by passing messages between images," Chertkov et al., PNAS (2010) 107: 7663–7668 [*chertkov zdeborova particle tracking messages 2010.pdf*]

Assignment

- Read the Chenouard *et al.* paper and select some of its references (coordinate with others). Describe one or more linkage algorithms to the rest of the group.

Module X Image Registration and Template Matching

How can we align images of the same object that may be distorted or rotated relative to each other? How can we find objects in images by template matching?.

COMMENTS TO MYSELF: This is neither as interesting nor as useful as most of the other topics. We covered it in 2014, but not in other years. The assignments need to be fleshed out. Maybe detecting fraud in images?

Readings

- Gonzalez, Woods, and Eddins, Chapter 6.
- "Geometric Transformation, Spatial Referencing, and Image Registration" in MATLAB's documentation.
- (if useful) "Fitzpatrick Hill Maurer Image Registration ch8.pdf"
- On Template Matching: Girod, Handout #9, 10 ("9-TemplateMatching_16x9.pdf")
- On Template Matching: Gonzalez, Woods, and Eddins, Chapter 13. This gets complicated quickly, but the key parts are near the beginning – e.g. using cross-correlation to match templates to images.

Assignment

- Find / take some images and apply image registration methods to them; follow along with Chapter 6 of Gonzalez, Woods, and Eddins.

- Take overlapping light sheet microscope images of gut regions and overlap them, interpolating intensity in some way. [Should note for next time: load these as an array, and note that the true scale is far less than 2^16.]
- Find a picture with some repeating pattern (e.g. widows in a building). Use cross-correlation with some template you create to find all the repeating elements.

Module X De-noising

Can we remove noise from images? How? Why?

COMMENTS TO MYSELF: We've always skipped this. It's interesting, but not as important as other topics. There are interesting questions about whether de-noising is pointless or not.

Readings

- Boulanger et al., "Patch-based non-local functional for denoising fluorescence microscopy image sequences"
- Old papers…

Assignment

Write code to implement this.