Biological Imaging and Analysis: Assignment 1

University of Cambridge

Henrik Åhl

May 6, 2017

Preface

This is an assignment report in connection to the *Biological Imaging and Analysis* module in the Computational Biology course at the University of Cambridge, Easter term 2017. All related code is as of May 6, 2017 available through a Github repository by contacting hpa22@cam.ac.uk.

1: Calculating the volume of a mouse embryo

The applied algorithm consists of applying in succession a range of tools in order to improve upon the analysis. The first things we do is to prune off the last 10 slices of our stack, since these appear to only contain background noise and not any direct cell matter *per se*. It also makes to stack slightly less asymmetric, as the first slice begins in middle of a layer of cells. With this more representative stack, we for all the subsequent applied tools use the middle slice, i.e. slice 50, as the baseline. As this is approximately in the middle of the tissue and ought to be fairly representative.

We thereafter despeckle the image, which applies a median filter supposed over a 3×3 pixel surface, with the intention of removing salt-and-pepper noise, i.e. single, individual bright or dark spots. The median filter itself simply takes out the median of the nine cells and sets all of the cells to this value. In order to smoothen the result, we thereafter apply a simple gaussian filter with $\sigma = 2$, which produces a gaussian kernel, or convolution matrix, in accordance to the equation

$$g(x, y) = \frac{1}{\sigma\sqrt{2\pi}} \exp^{-\frac{x^2+y^2}{2\sigma^2}}$$

, where σ is the rate of decay for the kernel. Every pixel is then replaced with the sum of the kernel elements multiplied correspondingly with the pixel's vicinity, with the kernel centered on the pixel [1].

x and y being the cartesian distances to the origion. σ the standard deviation of the change. Since we thereafter want to find the maxima of our cells in order to identify them using the `3D Find Maxima` function, we apply a maximum filter, which effectively is analogous to the median filter. In some of our cases we want to reduce the amount of noise in the image further, and therefore apply a `Remove Outliers` filter of radius 10 and threshold 0, which replaces pixels with the median of surrounding pixels if it deviates by more than the defined threshold. In other words, we filter out large chunks where cells have merged and become indistinguishable, as well as aberrant spots outside of the embryo [1].

The resulting image is then thresholded at slice 50 with either the Li or the Otsu method. Li's method minimises the cross-entropy between the threshold and the original image, whereas Otsu's method minimises the weighted variance between the thresholded and the original. In the case of Li's method, the cross-entropy takes the form of

$$\eta(t) = -m_{1fore}(t) \log \frac{m_{1fore}(t)}{m_{0fore}(t)} - m_{1back}(t) \log \frac{m_{1back}(t)}{m_{0back}(t)},$$

where the m's are the zeroth and first order moments of the foreground and background parts of the image respectively [2].

Otsu's method can be described as minimising the function

$$\sigma^2 = W_{back}\sigma^2_{back} + W_{fore}\sigma^2_{fore},$$

where the weights correspond to the normalised frequencies of the foreground and background intensities, and the σs simply the standard deviations [3].

In figs. 1 and 2 we see the inferred maxima from our approaches, with the parameter settings described in table 1. The idea here is of course that cell maxima should correspond to the cells themselves, although as we can see, the inferred number of cells vary wildly between the analyses. Comparing this to figs. 3 and 4,

which are the images on which the `Find Maxima` functio has been applied, we see that they do not differ significantly from each other visually. This naturally raises the importance of post-simulation analysis, with double-checking where the identified maxima are, and whether they correspond to actual cells. We can note some interesting features of our applied tools, however. For example, applying the `Remove Outliers filter` has indeed helped with misclassifying spots outside of the embryo as cells, as the differences between figures $A - D$ and $E - G$ shows. It is also not a significant factor in bringing down the number of inferred cells, which indeed is governed primarily by the assumed maxima radius, which we have exemplified by setting a lower value in one dimension. A major part of the analysis therefore comes down to accurately estimating an average cell size. Since we do have some visual artifacts making it hard to measure cell size in the z direction, the perhaps most accurate approach would be to assume spherical cells. When we do so, we arrive at ca 10000 cells, which appears visually reasonable.

Table 1: Table description of the different simulations and their corresponding input parameters and results, using a blob size of 17 μm.

Figure	*RO radius*	*Maxima radius*	*Gaussian* σ	*xy*	*z*	*No. maxima, Li*	*No. maxima, Otsu*
A	N/A	6	2	6	1.5	28442	19900
B	N/A	6	N/A	6	1.5	12754	10627
C	N/A	6	2	6	6.0	6917	5238
D	N/A	6	N/A	6	6.0	4465	4022
E	10	6	2	6	1.5	24758	18254
F	10	6	N/A	6	1.5	11393	9678
G	10	6	2	6	6.0	5247	4320
H	10	6	N/A	6	6.0	3545	3259

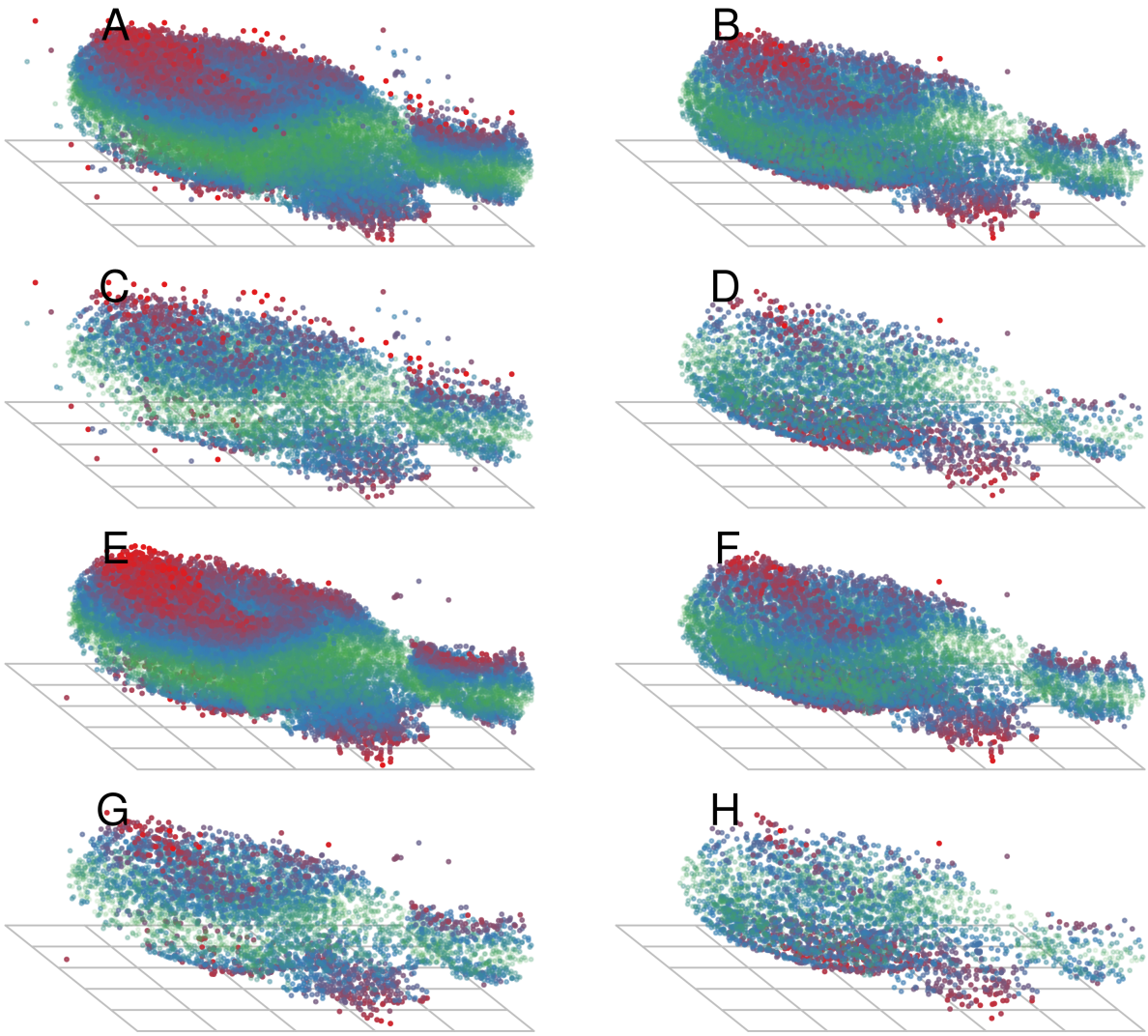

Figure 1: Identified maxima as produced by the Li thresholding method.

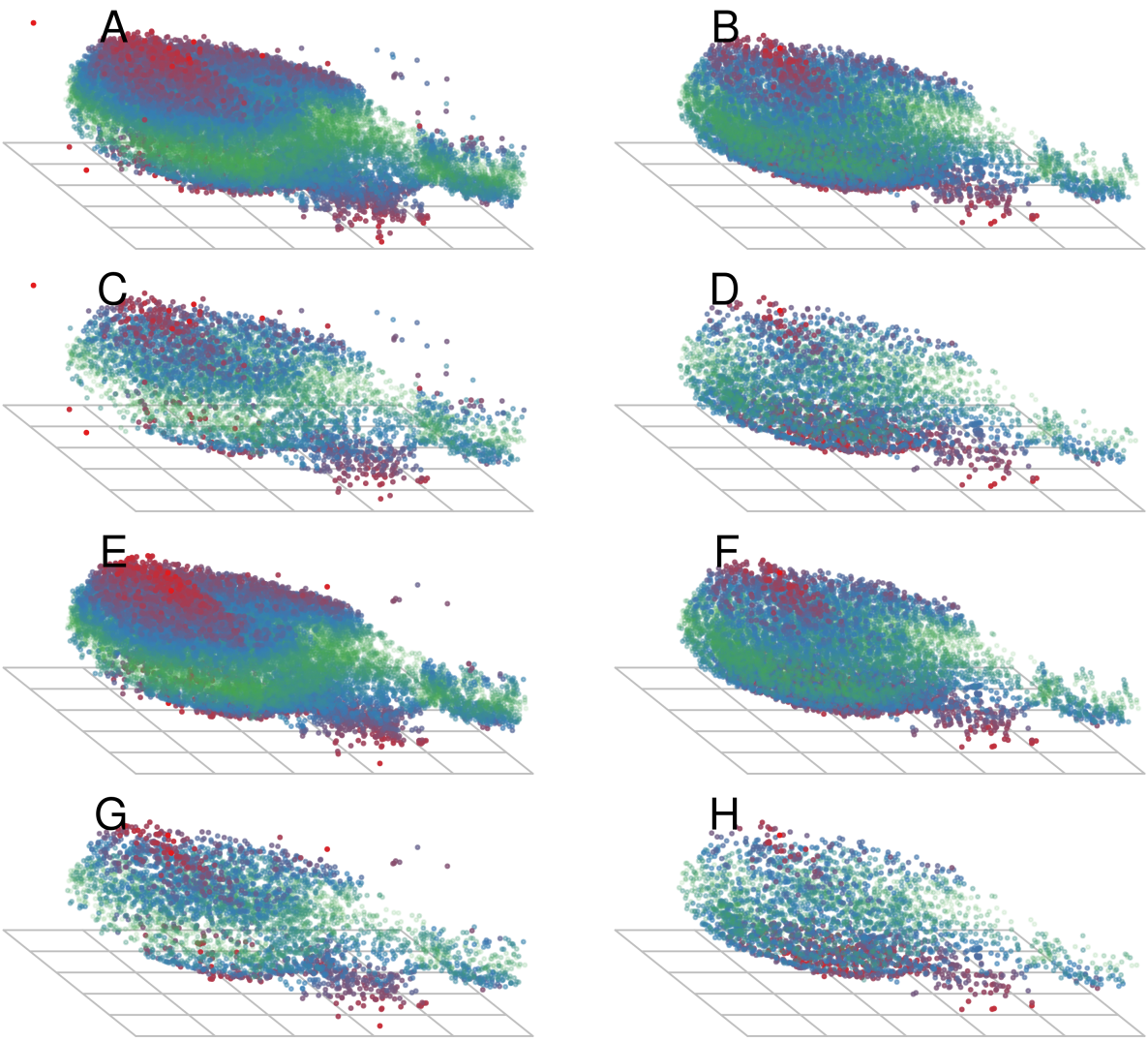

Figure 2: Identified maxima as produced by the Otsu thresholding method.

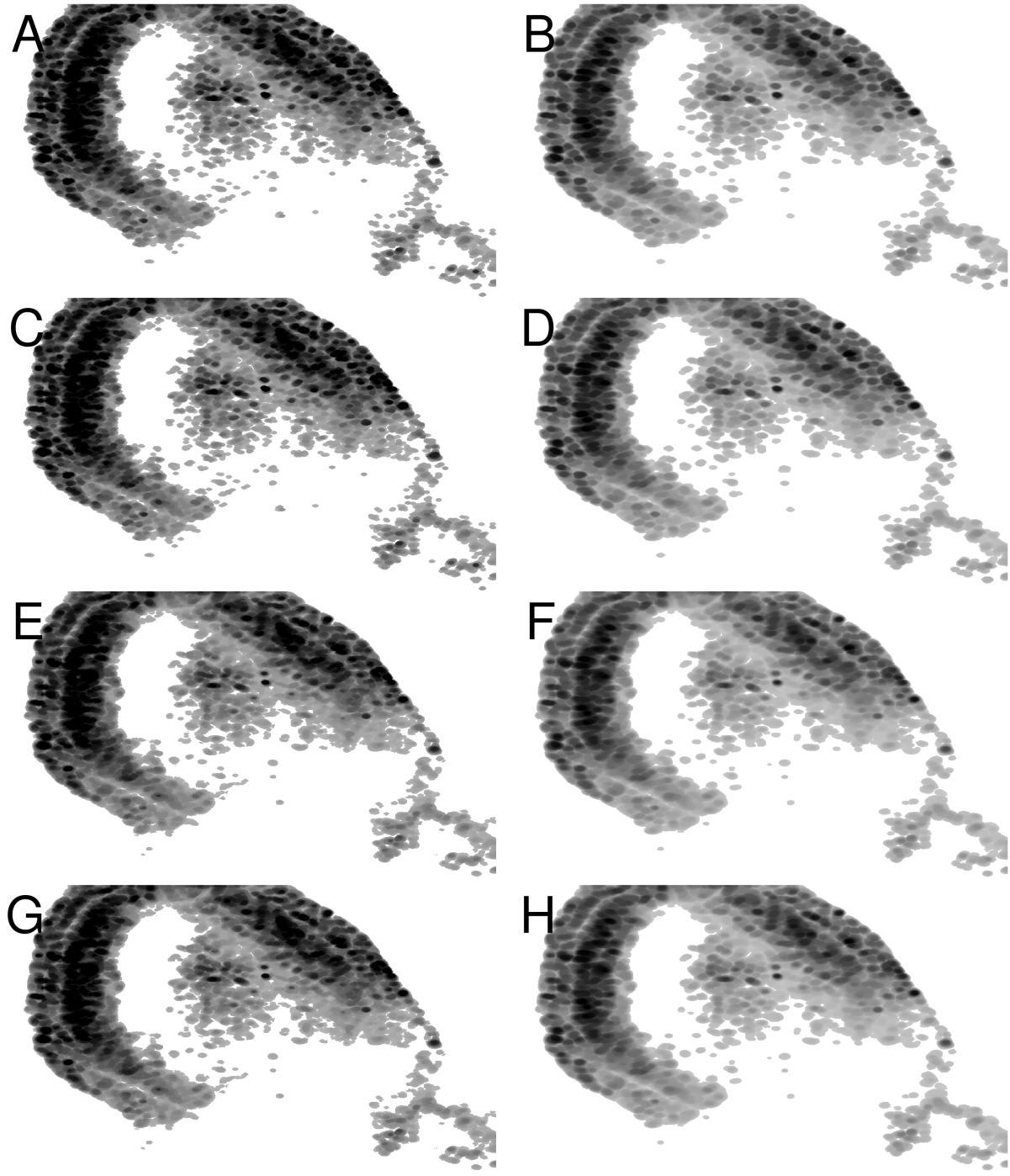

Figure 3: Li threshold. Thresholded and multiplied filtered image. Darker dots represent higher intensities.

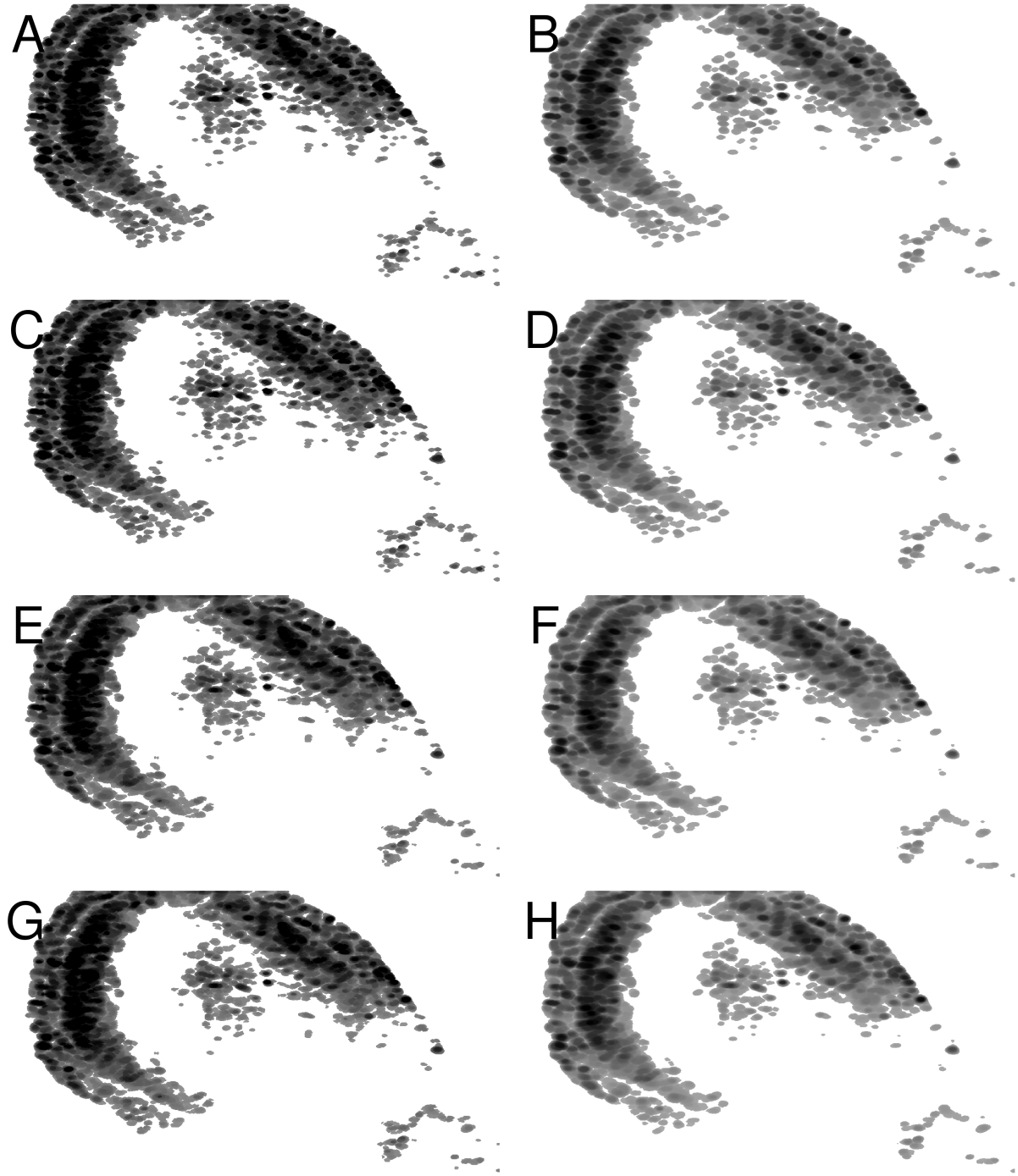

Figure 4: Otsu threshold. Thresholded and multiplied filtered image. Darker dots represent higher intensities.

When we disregard counting the cells directly through the measured maxima and instead try to infer the number of cells through calculation of the volume, we get the results seen in fig. 5. Here we used the same thresholded images as previously, but instead of modifying our processed image with it, we calculate the volume occupied by the thresholded chunk. As we clearly can see, $Li's$ method is less generous with which values are included when performing the threshold, as the volume is consistently smaller than for Otsus's method. The choice of method is therefore clearly very important when doing such an approximation, and we can correspondingly also expect large variances in our estimate of the number of cells from this. Being very rough, we can get the number of cells from the volume by estimating the volume of a single cell, and simply dividing the volume calculated from the thresholded image by this. Doing so is precisely what gives us the cell number estimate in fig. 5, where we have assumed a cell to be sperical with a radius of 6 pixels, which was concluded by manually measuring a set of cells. While this approach is naturally very inexact, it gives us an idea of the order of magnitude of the total number, which appears to be on the order of 10000.

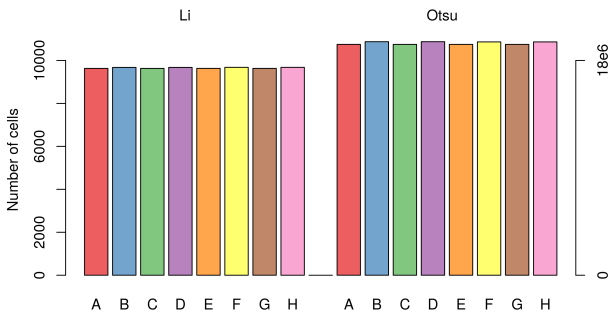

Figure 5: Otsu threshold. Thresholded and multiplied filtered image. Darker dots represent higher intensities.

In both of our attempts, we must however note that we are working with DAPI staining, which binds to the nuclear envelope, and not to the cell wall itself. We are therefore bound to underestimate the volume by default.

2: Tracking cells using TrackMate

In our second assignment we track a set of Histone 2B expressing cells using epifluorescence microscopy time-series data. Before initialising the tracking, we modify our images brightness and contrast significantly, in order to identify cells not visible at first glance. To safety check, we perform manual counting on the inital frame, where our method is indeed able to locate all 48 cells there from the beginning. The figure corresponding to this is seen in fig. 9. Note in particular the dark spots appearing in several places on the picture. Most of these, if not all, correspond to imaging artifacts, such as stains on the microscope lens or the likes. For several attempts to track our cells, these spots are wrongly identified as cells themselves, and we therefore choose to remove these with a twice applied `Remove Outliers` filtering. Similarly, due to the graininess of the image, we also do this once for white spots with a radius of 1.

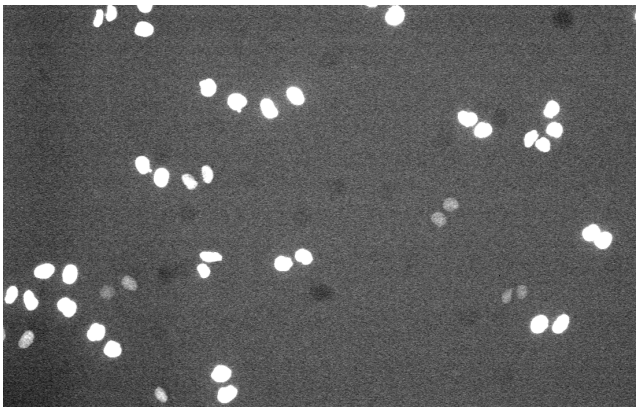

Figure 6: Image showing the initial frame after modifying contrasts. Note in particular the two dim cells in the bottom right corner, as these prove to be particularly hard to identify throughout. Also note the black stains, which correspond to imaging artifacts.

Using TrackMate, we perform the built-in automated median filtering without thresholding, and set the estimated blob diameter to 15 μm. This is an under-estimate, but done in order to not lose track of cells undergoing mitosis, as well as cells who give of low fluorescence. Since we work with nuclear staining, the identified objects occasionally vary significantly in size between frames. In particular during mitosis we can see how the chromosomes align, as this causes the cells to occur only as a thin, elongated stripes.

After identification of potential cells to track through a Laplacian of Gaussians (LoG) segmentation, which transforms all the pixels using the LoG equation

$$LoG(x,y) = -\frac{1}{4\pi\sigma}\left(1 - \frac{x^2+y^2}{2\sigma^2}\right)\exp^{-\frac{x^2+y^2}{2\sigma^2}},$$

effectively enhancing the outline of the cell. From this

we filter based on inferred quality, which is a predefined metric used by the *LoG* kernel precisely for blob detection.

The cells are tracked using the *LAP tracker*, which is supposed to be ideal for particles undergoing brownian motion, as we assume ours are. Since our images appear to allow for some jumping of our cells between frames, it occasionally happens that a cell division is mischaracterised as a new cell occuring from elsewhere. However, this appears to happen mostly in fringe cases, such as for the top-central, massive blob, which is a hard enough classification as is. We nevertheless choose to increase the frame-frame linkage to a distance of 20 μm, as well as set the splitting distance to 30 μm. Visually, this appears to give us better results. We also increase the maximum frame gap to 4, as cells sometimes merge in such a way that the segmenter are unable to recognise them individually. Due to this, it sometimes happens that cells are somethimes inferred to be part of an incorrect track; however, this seems like an unavoidable issue that cannot be resolved without fine-tuning cell trajectories manually. In addition, it does not affect our results, as we filter out tracks with disappearing members.

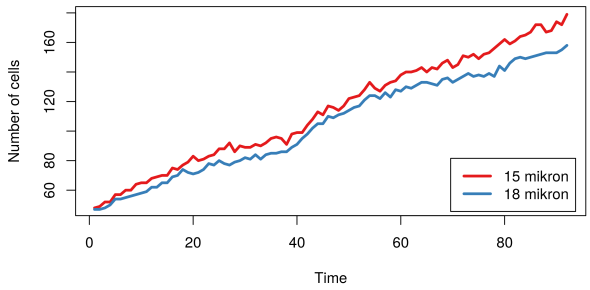

Figure 7: Number of cells identified in total, as time progresses. We note a linear in our time span, but note that we in a longer time-series would likely observe exponential growth. The two lines signify simulations using TrackMate with different estimated blob sizes. With a smaller estimate, the sensitivity is higher, and more cells are able to be identifies. However, this comes at the cost of losing more tracks when filtering, due to more cells dropping in and out during the time course.

Investigating the number of cell division events can be seen in table 2, where denoted filtering methods have been applied in the calculations. As we can see, when not filtering at all, and utilize all 48 tracks, the mean is still within the range of the pruned answers. In particular, when filtering out cells that fall within 20 pixels of the boundaries (ca 31 μm), we achieve a result very similar to the ones where we filter away all the tracks where any cell at any point disappear.

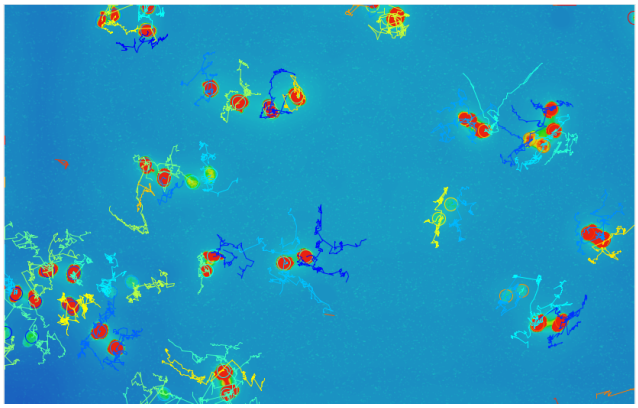

Figure 8: Cells and subsequenct tracks at the first frame, using an estimated blob size of 15 μm.

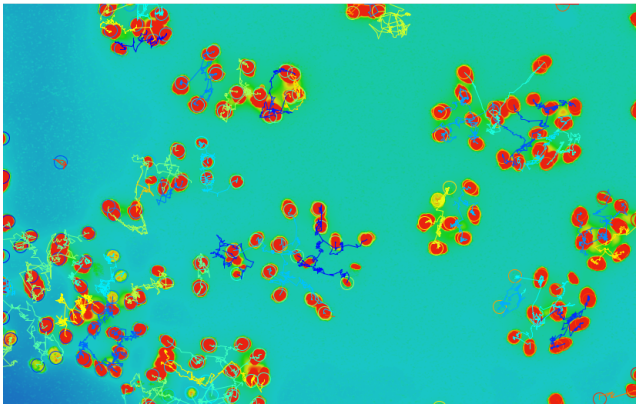

Figure 9: Cells and tracks at the last frame. Occasional new tracks appear, as can be seen on the lonely, scattered lines appearing at a distance from the other clusters.

Naturally, our method suffers from several drawbacks that are all hard to circumvent. The perhaps msot obvious one is the requirement of cells to be within view throughout the simulation, which is what causes the most drastic loss in tracks when filtering. In several cases, cells simply fall outside the x-y boundaries, where exclusion is more reasonable. However, when a cell drops out of view in the z-dimension, it is not as clear whether we should exclude it or not, since there is often some sort of visual cue that the cell is still present, even though the segmentation is not able to identify it. A similar problem occurs when cells come too close to each other in such a way that their intensities overlap such that they become indistinguishable from each other, and we are again forced to exclude the whole track from the analysis. In addition, the same thing happens during mitosis, when

Table 2: Table description of the different simulations and their corresponding input parameters and results.

Filter	*Number of tracks*	*Mean*	*SE*
None	48	2.02	0.25
Edges	35	2.31	0.28
Disappearance	17	2.71	0.44

the chromosomes align and become thin, as we are attempting to identify the targets as approximately spherical (circular).

There is also a problem with identifying what cells belong to a certain track due to cell division events that cause one of the offspring to occur in a far-away position. The risk is that this new cell, which is supposed to be considered a part of the same track as the other cell, is simply regarded as a completely new track on its own. Having a higher allowed split distance would work as a counter to this, but would instead suffer from misidentifying cells coming in from outside the field of vision as cell division events, which we definitely do not want. Similarly, allowing for a higher number of frames for where a cell can be absent while still considering it to be part of the track when it reappears introduces the problem with replacing a cell with another one which just so happens to be in a position where the factual cell could have been.

However, despite the many problems with this general approach, we see merit in many parts of applying it as a type of preprocessing. In several cases, TrackMate is able to identify candidate cells which would be harder to manually justify. We also get the benefit of consistency – whereas a human doing manual tracking would primarily evaluate cells based on their visual appearance, TrackMate only considers the hard numbers, which makes its classification well-defined and concise. There is naturally also the benefit of using the automated approach as primarily complementary, and simply pruning the results from our algorithms manually afterwards. Still, while using software to do the main body of the tracking is indeed automated, there are high amounts of manual tweaking which has to be done beforehand, in addition to the post-tracking adjustments. Many of these settings are fairly arbitrary, and it is hard from the viewpoint of the user to be able to determine what makes the algorithm perform optimally. Often the preprocessing is done based on what is perceived as the most visually compelling, which is not necessarily what is optimal for the algorithm. In other words, while there are many helpful results coming out of the tracking as done here, there are many unresolved questions of how to improve on the results significantly. Still, the analysis itself can often be adapted to allow for insightful results; investigating the number of cell division events only requires us to be able to identify when a cell is undergoing mitosis, and only makes up a small fraction of all the events taking place in the time-series, which in addition also makes them significantly easier to manually correct in retrospect.

Track files can be downloaded at https://www.dropbox.com/s/2xpof5hhz6yid0n/N2DL-HeLa_Tracks.xml?dl=0 (15 μm) and https://www.dropbox.com/s/evx95zriehyb5fr/N2_Tracks.xml?dl=0 (18 μm).

References

[1] Imagej methods, 2017. URL https://imagej.nih.gov/ij/docs/guide/user-guide.pdf.

[2] CH Li and Peter Kwong-Shun Tam. An iterative algorithm for minimum cross entropy thresholding. *Pattern Recognition Letters*, 19(8):771–776, 1998.

[3] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *IEEE transactions on systems, man, and cybernetics*, 9(1):62–66, 1979.

A Code

```
// ImageJ macro for command-line usage. Remember to change the paths if
// necessary. Usage:
//
// path-to/ImageJ-linux64 --allow-multiple --headless -macro path-to/macro.ijm \
//     'Threshold_method useOutliers Gaussian_sigma \
//     Maximum_filter_radius XY_radius Z_radius useGaussian'

path="/home/hpa22/project_out/"
filelist = getArgument();
file = split(filelist,' ');
argssuff = file[0]+"_"+file[1]+"_"+file[2]+"_"+file[3]+"_"+file[4]+"_"+file[5] +"_"+ file[6];

method=file[0];
open("project_data/Embryo.ome.tif");
run("Duplicate...", "duplicate range=1-100");
run("Properties...", "channels=1 slices=100 frames=1 unit=pixel pixel_width=0.5 pixel_height
    =0.5 voxel_depth=1.98 global");
run("Duplicate...", "duplicate range=1-100");
Stack.getDimensions(width, height, channels, slices, frames);
Stack.setSlice(50);

// Applying filters
print("Applying filters...");
run("Enhance Contrast", "saturated=0.35");
run("Apply LUT", "stack");
if(file[6] == 1){
  run("Gaussian Blur...", "sigma="+file[2]+" stack");
}
run("Maximum...", "radius="+file[3]+" stack");
//run("Despeckle", "stack");
if(file[1] == 1){
  run("Remove Outliers...", "radius=10 threshold=0 which=Bright stack");
}
rename("Embryo.ome-2.tif");
run("Duplicate...", "duplicate range=1-100");

// Set threshold and apply mask
print("Setting threshold and applying mask...");
setAutoThreshold(""+method+" dark");
run("Convert to Mask", "method="+method+" background=Dark");
saveAs("Embryo.ome-3.tif",path+argssuff+".tif");
rename(path+argssuff+".tif");
run("Divide...", "value=255 stack");
imageCalculator("Multiply create stack", "Embryo.ome-2.tif", path+argssuff+".tif");
rename("Result of Embryo.ome-2.tif");
selectWindow("Result of Embryo.ome-2.tif");
//run("8-bit");
run("Invert LUT");
saveAs("Result of Embryo.ome-2.tif", path+argssuff+"_multiplied.tif");

// Find maxima
print("Doing 3D Maxima finding");
xyrad=file[4];
zrad=file[5];
nnoise=0;

side_chunks = 1;
for(i=0; i<side_chunks; i++){
  for(j=0; j<side_chunks; j++){
  print("(" + i + ", " + j + ")" + "...");
  selectWindow("Result of Embryo.ome-2.tif");
  makeRectangle(i*width/side_chunks, j*height/side_chunks, width/side_chunks, height/side_
      chunks);
  run("3D Maxima Finder", "radiusxy="+xyrad+" radiusz="+zrad+" noise="+nnoise);
  saveAs("Results", path+argssuff+"_res"+i+j+".dat");
```

```
}

exit;
```

../code/process_embryo.ijm

```
#!/usr/bin/env Rscript
# Plot all the maxima we have inferred from our Find Maxima approach
setwd("~/compbio/src/assignments/bia1/code/")
library(RColorBrewer)
library(scales) #imports alpha
library(stats)
library(plot3D)
palette(brewer.pal(n = 8, name = "Set1"))
color = brewer.pal(n = 8, name = "Set1")
lw.s = 3

# Load files and modify colours
files = list.files(path="~/project_out/", pattern="res", full.names=F)
alphs = c(seq(1,.1, length.out=50), rev(seq(1,.1, length.out=50)))
colors = alpha(colorRampPalette(color[c(1,2,3,2,1)])(100), alphs)

# Do Li
par(mfrow=c(4,2),mar = c(0,0,0,0), oma = c(0, 0, 0, 0) + 0.0, mai = c(.0, .0, .0, .0))
count = 1
for (file in files[1:8]){
  data = read.csv(paste0("~/project_out/", file), sep = "\t")
  colnames(data) = c("Index", "x", "y", "z")
  cols = cut(data$z, 100) # Colour by z-value
  scatterplot3d(x=data$x*.5, y=data$y*.5, z=data$z*1.981818, color=colors[cols], xlab="",
      ylab="", zlab="", axis = F, asp = .8, pch = 20, angle=120, main = "", mar = c(0,0,0,0))
  mtext(side=2, LETTERS[count], line = -8, las = 1, at=7, cex = 1.5)
  count = count + 1
}

# Do Otsu
for (file in files[9:16]){
  data = read.csv(paste0("~/project_out/", file), sep = "\t")
  colnames(data) = c("Index", "x", "y", "z")
  cols = cut(data$z, 100)
  scatterplot3d(x=data$x*.5, y=data$y*.5, z=data$z*1.981818, color=colors[cols], xlab="",
      ylab="", zlab="", axis = F, asp = .8, pch = 20, angle=120, main = "", mar = c(0,0,0,0))
  mtext(side=2, LETTERS[count], line = -8, las = 1, at=7, cex = 2)
  count = count + 1
}
```

../code/plot_maxima.R

```
#!/usr/bin/env Rscript
# Script to extract and plot the images resulting from
# multiplying the processed and the thresholded images.
setwd("~/compbio/src/assignments/bia1/code/")
library(RColorBrewer)
library(scales) #imports alpha
library(stats)
library(tiff)
library(raster)
palette(brewer.pal(n = 8, name = "Set1"))

# Adapt this accordingly
files = list.files(path="~/project_out", pattern="multiplied", full.names=T)

# Li
count = 1
par(mfrow=c(4,2),mar = c(0,0,0,0), oma = c(1, 1, 1, 1) + 0.0, mai = c(.0, .0, .0, .0))
for(file in files[1:8]){
  f = readTIFF(file, all=T, native = T)[[50]] # Take slice 50
```

```
  image(f, axes=F, col = c("black", "white"))
  mtext(side=2, LETTERS[count], line = -3, las = 1, cex = 2, at = .9)
  count = count + 1
}

# Otsu
count = 1
par(mfrow=c(4,2),mar = c(0,0,0,0), oma = c(1, 1, 1, 1) + 0.0, mai = c(.0, .0, .0, .0))
for(file in files[9:16]){
  f = readTIFF(file, all=T, native = T)[[50]]
  image(f, axes=F, col = c("black", "white"))
  mtext(side=2, LETTERS[count], line = -3, las = 1, cex = 2, at = .9)
  count = count + 1
}
```

../code/plot_tifs.R

```
#!/usr/bin/env Rscript
# Script to plot the total cell volume and the inferred
# cell size, as approximated by
library(RColorBrewer)
library(scales) #imports alpha
library(stats)
library(plot3D)
palette(brewer.pal(n = 8, name = "Set1"))
color = brewer.pal(n = 8, name = "Set1")
lw.s = 3

# Read in files
files = list.files(path="~/project_out/", pattern="voxels", full.names=F)

# Get the total area
areas = vector("numeric", length=length(files))
count = 1
for(file in files){
  data = read.csv(paste0("~/project_out/", file), sep = ",")
  areas[count] = sum(data$Area)
  count = count + 1
}

# Get the right cell volume
voxel.volume = 4/3*pi*6**3 * 0.5**2 * 1.981818
xy.z.discrepancy = 4

# Plot!
par(mfrow=c(1,1),mar=c(5,5,5,5))
barplot(c(areas[1:8], 0, areas[9:16])/(voxel.volume*xy.z.discrepancy),
        col = alpha(1:9, .7), names.arg = c(LETTERS[1:8], "", LETTERS[1:8]), yaxt ="n")
mtext("Li",    side=3, line = 1, at = 4.50, cex = 1)
mtext("Otsu", side=3, line = 1, at = 15.5, cex = 1)
axis(2)
axis(4, labels = c("0", "18e6"), at = c(0, 20000))
mtext("Number of cells", side=2, las = 3, line = 2.5)
mtext(paste0("Volume [mikron]"), side=4, las = 3, line = 2.5)
```

../code/estimate_and_plot_area.R

```
//run("Brightness/Contrast...");
run("Enhance Contrast", "saturated=0.35");
setMinAndMax(32935, 33265);
run("Apply LUT", "stack");
run("Remove Outliers...", "radius=30 threshold=0 which=Dark stack");
run("Remove Outliers...", "radius=30 threshold=0 which=Dark stack");
run("Remove Outliers...", "radius=1 threshold=0 which=Bright stack");
rename("N2");
saveAs("N2", "/home/henrik/n2_spotremoved.tif");
//run("Fire");
```

```
//run("TrackMate");
```

../code/preprocess_tracking.ijm

```
#!/usr/bin/env Rscript
# Do the track analysis
setwd("~/compbio/src/assignments/bia1/code/")
library(RColorBrewer)
library(scales) #imports alpha
palette(brewer.pal(n = 8, name = "Set1"))
lw.s = 3

# Read in data
# 14 if lower & so
track.file = "../data/All Spots statistics14.csv"
data = read.csv(track.file, sep = ",")
data = data[,c(2,4,5,6,7,9,14,15,16,17,18)] # Take away columns we don't want

##############################################
### Plot spots / time
##############################################
no.cells.per.time = unname(table(data[,"POSITION_T"]))
plot(1:92, no.cells.per.time, type = "l", yaxt = "l", ylab = "Number of cells", xlab = "Time"
    , lwd = lw.s, col = 1)
axis(2)

##############################################
### Find the tracks and filter accordinly
##############################################
no.tracks = length(unique(data[,"TRACK_ID"])) # How many tracks do we have?
cat("no tracks throughout: ", no.tracks, "\n")

# Take away the ones that weren't there to begin with, as well as the ones
# marked as "None" -- these correspond to crappy, short-lived tracks that
# shouldn't be accounted for.
init.filter = which(data$POSITION_T == 0)
ids = unique(data[init.filter, "TRACK_ID"]) # Retrieve the actual tracks present at t = 0.
ids = ids[which(ids != "None")] # Remove Nones.
no.tracks = length(unique(data[which(data$TRACK_ID %in% ids),"TRACK_ID"])) # How many tracks
    do we have?
cat("no tracks in the beginning: ", no.tracks, "\n")

# Find all tracks where cells disappear during the course of the
# time series..
all.splits = c()
for(ii in ids) {
  cells.in.id = which(data$TRACK_ID == ii)
  # If faulty, remove and go to next track.
  if (length(unname(table(data[cells.in.id, "POSITION_T"])))) != 92 |
      any(diff(table(data[cells.in.id, "POSITION_T"])) < 0))
  {
    data = data[-cells.in.id, ]
    next
  }
  # Collect all the splits we have left
  all.splits = c(all.splits, sum(diff(table(data[cells.in.id, "POSITION_T"])))))
}

output = c(length(all.splits), mean(all.splits), sd(all.splits) / sqrt(length(all.splits)))

##############################################
# Redo analysis above, but allow for disappearances.
##############################################
# Find all tracks where cells disappear throughout
data = read.csv(track.file, sep = ",")
data = data[,c(2,4,5,6,7,9,14,15,16,17,18)]
colnames(data)[c(2, 4:6)] = c("id", "x","y","t")
all.splits = c()
```

```
63
64 for(ii in ids) {
65   cells.in.id = which(data$id == ii)
66   # How many splits based on how many are there in the end?
67   l = table(data[cells.in.id, "t"])
68   all.splits = c(all.splits, l[length(l)] - 1)
69 }
70 output = rbind(output, c(length(all.splits), mean(all.splits), sd(all.splits) / sqrt(length(
       all.splits))))
71
72 ##########################################################################
73 # Redo analysis, but only remove cells which fall outside of the edge
74 ##########################################################################
75 # Find all tracks where cells disappear throughout
76 data = read.csv(track.file, sep = ",")
77 data = data[, c(2, 4, 5, 6, 7, 9, 14, 15, 16, 17, 18)]
78 colnames(data)[c(2, 4:6)] = c("id", "x","y","t")
79 all.splits = c()
80 width = 1100
81 height = 700
82 pix.per.um = 1100/709.5 # ca 1.55 pixels / mikron
83 limit = 30 # pixels
84
85 for(ii in ids) {
86   cells.in.id = which(data$id == ii)
87   if(any(data[cells.in.id, "x"] < 0 + limit | data[cells.in.id, "x"] > width  - limit |
88          data[cells.in.id, "y"] < 0 + limit | data[cells.in.id, "y"] > height - limit)){
89     data = data[-cells.in.id, ]
90     next
91   }
92
93   # How many splits based on how many are there in the end?
94   all.splits = c(all.splits, (table(data[cells.in.id, "t"]))[length(table(data[cells.in.id, "
       t"]))] - 1)
95 }
96 output = rbind(output, c(length(all.splits), mean(all.splits), sd(all.splits) / sqrt(length(
       all.splits))))
97
98 output[c(2,1),] = output[c(1,2), ]
99 output[c(2,3),] = output[c(3,2), ]
100 print(output)
```

../code/analyse_tracks.R