PHY331 tutorial
Oct. 23rd 2019

TA: Chris Nunn
Office: MP415

cnunn@physics.utoronto.ca

*Please hand in Exercise 4.

**PS4 and Tutorial exercise 5 are due next week in tutorial, Oct 30th.

Review of upcoming schedule

- Today: Particle tracking 1, exercise 5.
- Oct 30th: Particle tracking 2.
- Nov 6th: No class/tutorial due to reading week.
- Nov 13th/20th/27th: Thermal motion lab/computer lab time.
- Dec 3rd: Review for final exam.

Particle tracking part 1: preparing for the thermal motion lab

Here is the link to the procedure we will follow:

https://www.physics.utoronto.ca/~phy224_324/experiments/thermal-motion/Thermal%20Motion.pdf

I will send out an email in the next few weeks letting you know your lab partners and on which week you are expected to be in the wet lab. I will be jumping back and forth between the microscopes and computer lab to help with your coding.

- Today: image processing
  - In the thermal motion lab we will be imaging fluorescent particles undergoing Brownian motion
  - The computer captures a series of images over time, and we need to be able to identify particles between frames to extract meaningful statistics
  - First we will discuss thresholding pixel intensity to identify clusters of bright pixels (beads).
  - Then discuss how clusters of bright pixels (beads) will be labeled so that we can identify them later

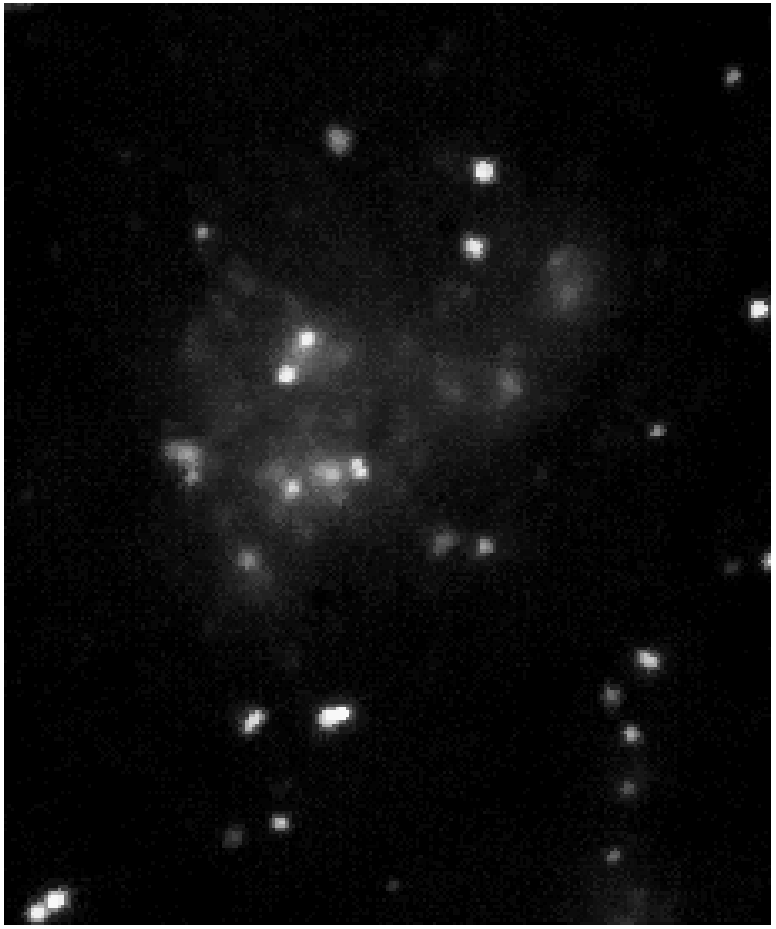
Image processing in microscopy

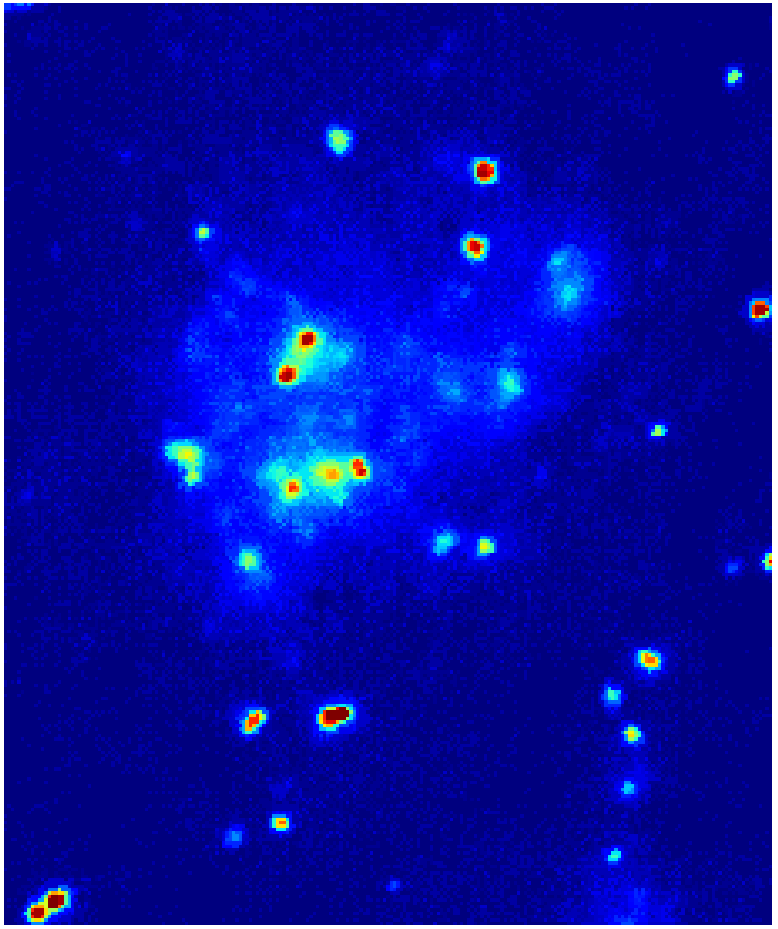
- Often the fluorescence of tagged objects (protein, DNA, virus...) is monitored, and confined within a region so that most movement is in 2D.
- A stack of images are taken and we then process the data to output the trajectories of the objects.
- There is some trickiness in getting “good” output...

Difficulties

- Background noise vs signal.
  - Only some objects are in the focal plane of the microscope, so not every signal is “bright enough”.
- Trajectories of the objects
  - They can move out of the frame/field of focus and then reappear at a later time.
- Objects are too close to each other
  - Then the fluorescence signal will appear contiguous and misidentified as a single object, but in fact they are not.

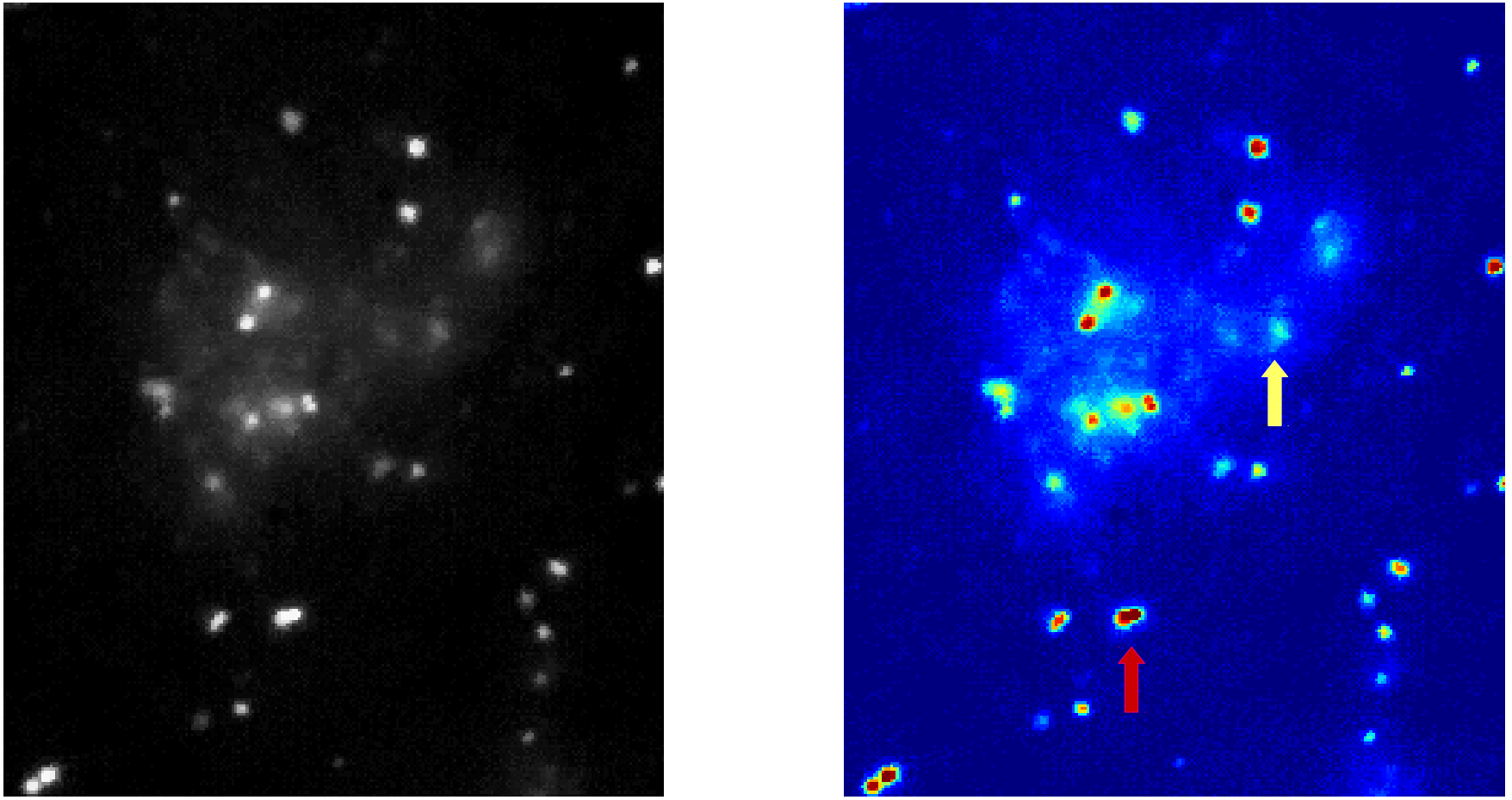
Transport of virus along cytoskeleton filaments





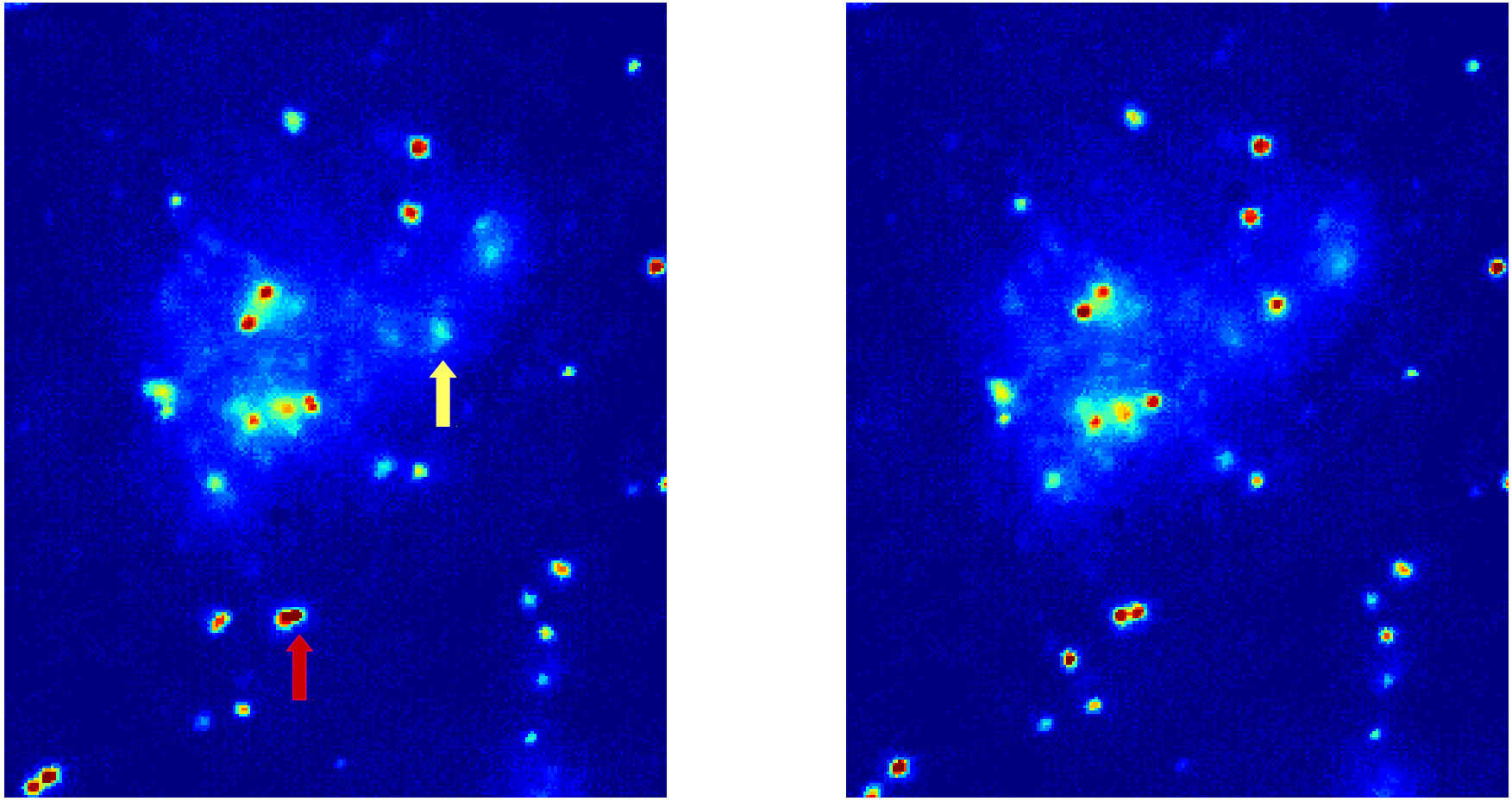
(Taken from http://mosaic.mpi-cbg.de/MosaicToolboxSuite/ParticleTracker.html)

Transport of virus along cytoskeleton filaments



T = 0 frame, right image shows the “sliced” 8-bit image plotted with a colour scheme. Red arrow: fluorescence spots that are too close to each other.

Transport of virus along cytoskeleton filaments



Right image: only 2 frames later, the fluorescence spot pointed at by the yellow arrow emerges in the focal plane and thus appears brighter than before.

- There are three main components of getting the processed output and trajectory data:

  1) Identify the bright pixels in the stack of images, apply a threshold to make the contrast as high as possible.

  2) Cluster the bright pixels and label each cluster within each frame of the images.

  3) Cluster the labels in time to produce the trajectories (i.e. identify the correlation between the clusters in successive frames). **Next week**.

Loading images

- Using matplotlib.image package we can work with image files (tif/png/jpg...etc.)
- To load an image:

```
import matplotlib.image as mpimg

fname = '/path_to_your_file/file_name.tif'

img = mpimg.imread(fname)
img = img[:,:,0]              #This line is required only for tif!!

plt.imshow(img)
plt.show()
```

Loading images cont’d

- imread() imports the image file as an 8-bit image. It is an NxM array (where N and M are the number of pixels in each direction)
- **In the case of tif**: each array is a list of 3 numbers, each represents the RGB pixel intensity.
- png format can have up to 4 numbers (RGB intensity + opacity).
- The line “img = img[:,:,0]” simply forms a NxM matrix with one element from the list – since if we import a BW image, all the RGB intensities are similar.

Thresholding

- With the img array, we then threshold its intensity values.
- “Simple” way: pick a global threshold pixel intensity T for every frame:

```
for i in range(0,N):
 for j in range(0,M):

  if img[i][j] > T:
   C[i][j] = 1
  else:
   C[i][j] = 0
```

- Note that there are better algorithmic ways to pick the “best” threshold (Otsu’s method).

Thresholded image

- We threshold an actual image taken from the thermal motion experiment.
- Unfortunately, this is not yet too helpful for tracking the individual particles yet...

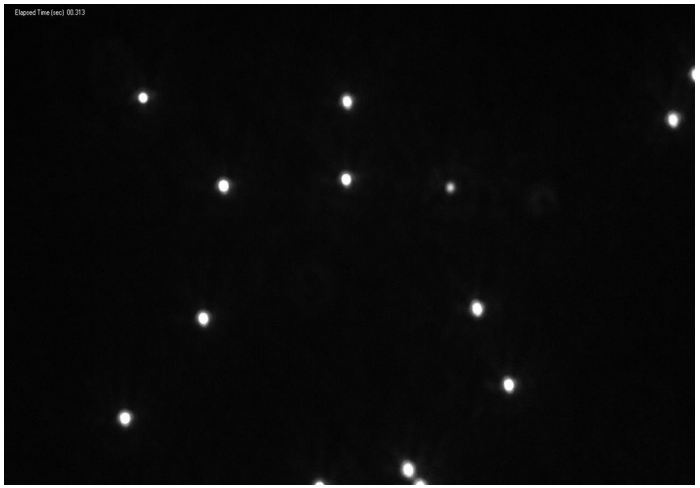
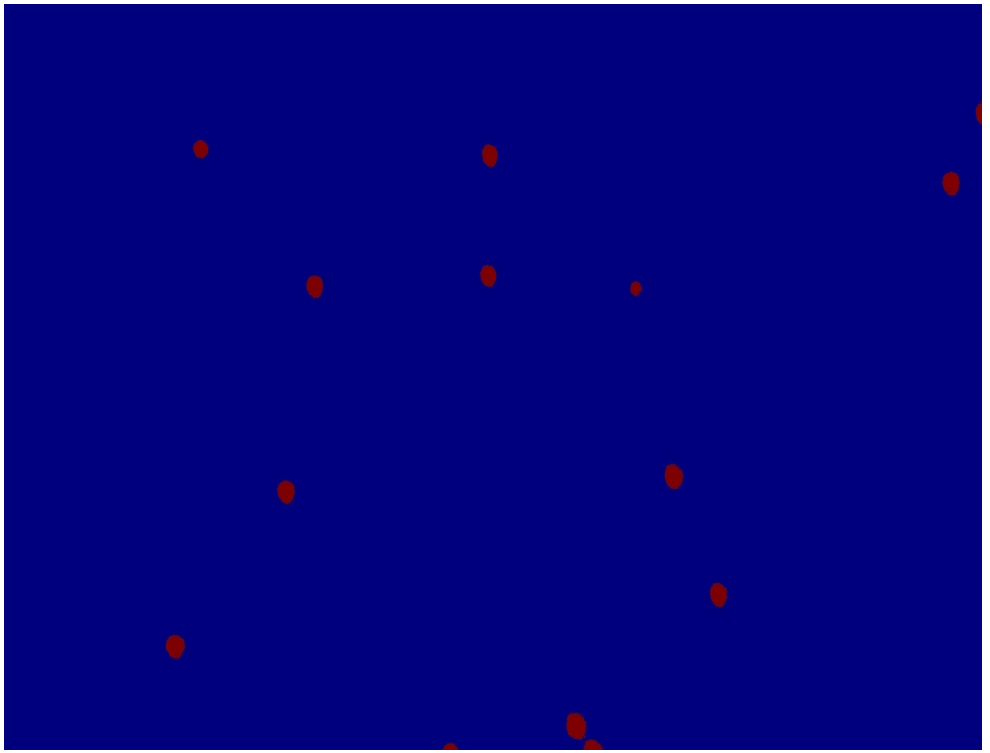
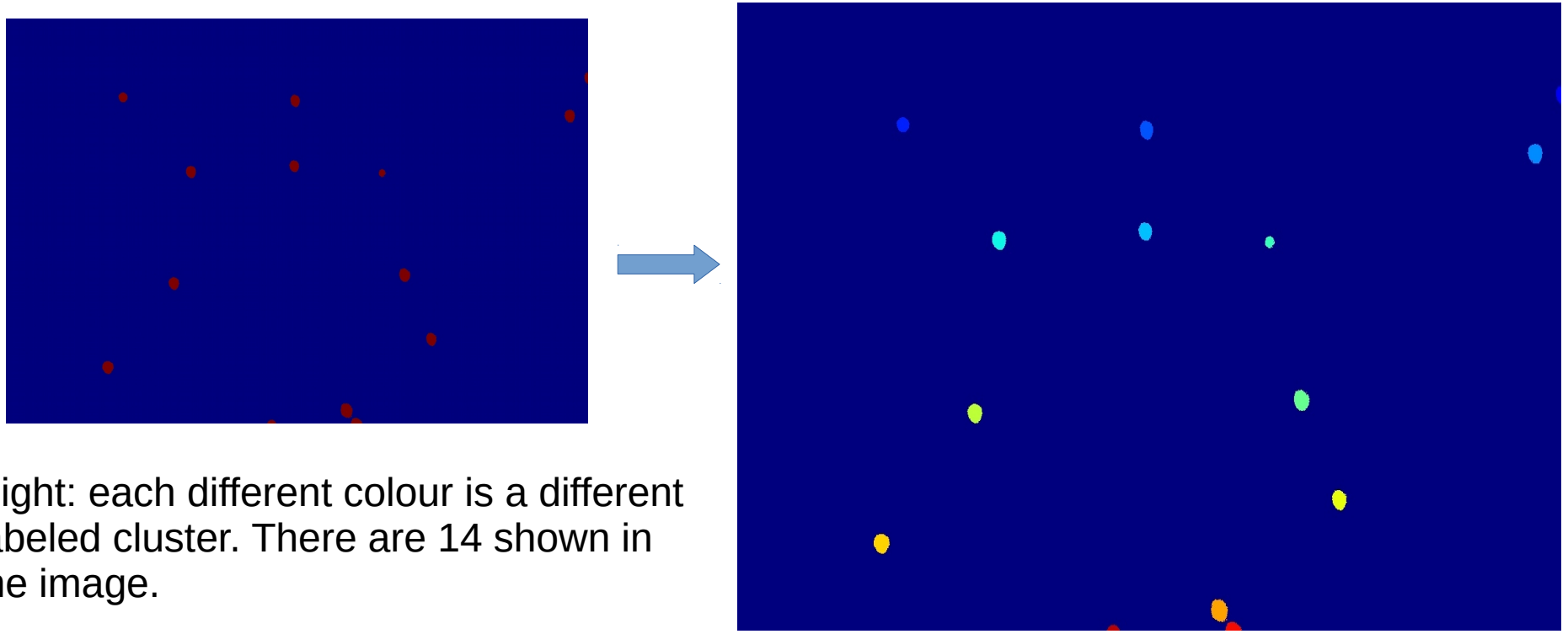


Image from thermal motion experiment



Labeling the clusters

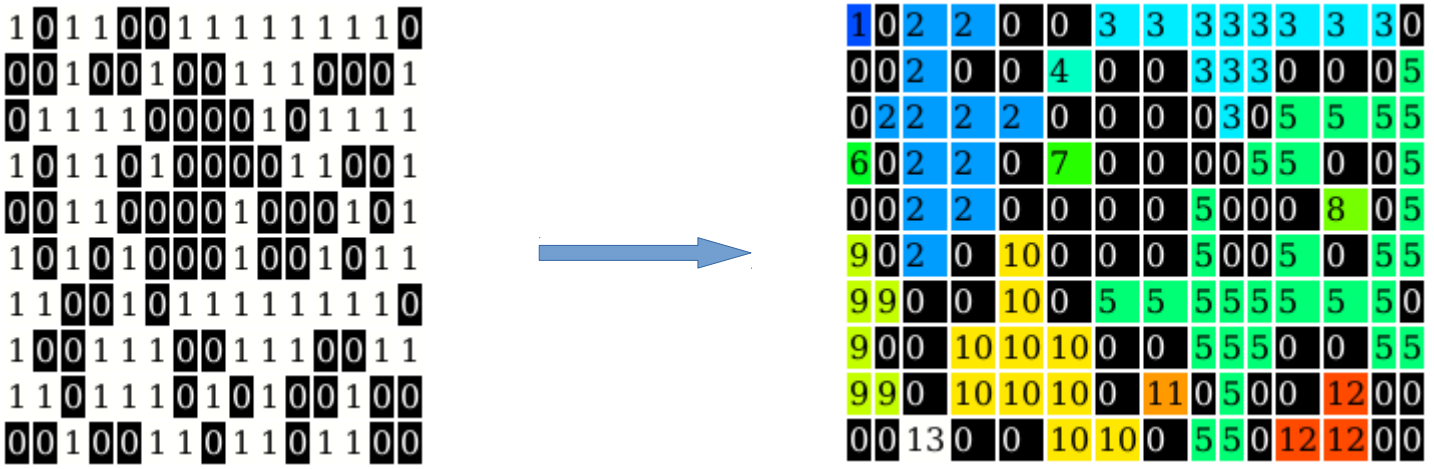
- The next thing we want to do, is to assign to each cluster of 1's (bright pixels) their own unique label, by the order in which the rows of the image are read.



Right: each different colour is a different labeled cluster. There are 14 shown in the image.

Hoshen-Kopelman algorithm

Check out:
https://www.ocf.berkeley.edu/~fricke/projects/hoshenkopelman/hoshenkopelman.html



Reads contiguous clusters of 1's and returns unique labels for each cluster. The code is given in cluster.py and has 3 auxiliary functions: uf_makeset(), uf_union() and uf_find().

The main function, hoshen_kopelman(), returns the number of clusters identified in each frame.

How HK works

- There are two parts to HK, the matrix is scanned twice: first to identify the clusters/number of labels, then the second time the new labels are assigned.
- When a pixel (i,j) is =1, we check its LEFT and UP neighbours, if both LEFT and UP are 0's, we call uf_makeset() to create a new label.

- If one of the LEFT/UP neighbours is nonzero, but the other is zero (i.e. the pixel (i,j) has only 1 neighbour), then the pixel itself gets the label of its neighbour.
- Third case: neither neighbours are empty, uf_union() is called to setup an equivalence class between the LEFT/UP neighbours (via the array label).
- Second scan of matrix: uf_find() is called and the equivalence class of labels are “collapsed”. Now each cluster has been relabeled correctly.

- Having unique labels on each cluster makes it easy to calculate its position (given by the centre of mass of all the pixels with the same label).

$$\vec{R}_{cm,i} = \frac{1}{N_i} \sum_{j=1}^{N_i} \vec{R}_j(i)$$

N_i - number of pixels with label i

$\vec{R}_j(i)$ - position of the j^{th} pixel which has label i

- “Radius” of the particle:

$$R_g^2 = \frac{1}{N_i} \sum_{j=1}^{N_i} [\vec{R_j}(i) - \vec{R}_{cm,i}]^2$$

“Radius of gyration”: similar in concept to HW1, Q3
...also you might have learned this in class when dealing with polymers.