A MATLAB toolbox for virus particle tracking

Ivo F. Sbalzarini[*]

May 28, 2007

Abstract

This document describes a set of MATLAB[1] scripts and functions developed to acquire virus particle trajectories from video microscopy recordings. Input data are video frames of fluorescently labeled moving particles whose size is below the resolution limit of the microscope (i.e. of moving point spread blobs). The individual frames of the video need to be available as image files. In each frame image, the software detects the locations of virus particles while trying to reject other bright spots such as secondary endocytic vesicles, dust or autofluorescence. The obtained locations in each frame are then optimally linked into trajectories over time.

ICoS Technical Report

[*]Computational Biophysics Lab, Institute of Computational Science, Department of Computer Science, ETH Zürich. (`http://www.cbl.ethz.ch`). E-mail: ivos@ethz.ch

[1]MATLAB is a trademark of The Mathworks, Inc.

1 Introduction

This toolbox was written in the course of a collaborative project between the institutes of computational science and biochemistry at ETH Zürich with the goal to automatically acquire large numbers of virus trajectories from digital video microscopy. The toolbox consists of a collection of MATLAB scripts and functions and makes use of MATLAB's Image Processing Toolbox. It was developed and tested in MATLAB 6.1.

The task is to extract the trajectories of moving bright blobs (observed point spread blobs of fluorescent sub-resolution particles) from frame images of a video recording. This is done in two steps. First, the particle positions are located in each movie frame. In a distributed environment or on a multi-processor machine this task can be accelerated by executing it in parallel (see section 6 for more information on how to run parallel jobs). As a second step, these lists of positions are linked into trajectories over time so as to maximize a certain measure of quality.

One might well ask why we developed *another* particle tracking software instead of using one of the many that are available for example from fluid dynamics (PTV measurements) robotics (machine vision, feature tracking) or biology (cell tracking, fluorescence speckle microscopy). The reason is that tracking virus particles seems to be a different problem than tracking fluid tracers, hands, faces, speckles or cells and therefore needs a different tracking algorithm. Some of the reasons for this are:

1. Virus particles are smaller than the resolution limit of the microscope. What one observes are thus point spread functions (blobs) whose shape is given by the optics of the microscope and whose size is given by the relative position of the virus particle to the focal plane. The shape of the object can therefore not be used to identify its orientation in space (and thus the expected direction of motion) as it is for example done in machine vision, hand- or face tracking or cell tracking.

2. Neighboring virus particles can exhibit completely different motion. Spatial correlation measures[2] such as the ones used in fluid dynamics are therefore inappropriate.

3. Virus particles can move in what seems to be a random walk. Spatial coherence measures requiring that the particles move along smooth curves can therefore not be used to solve the tracking problem. Any tracking algorithm from fluid dynamics, machine vision, cell tracking or fluorescence speckle microscopy will thus not solve the problem.

4. Virus particles can switch between a "random" and a "directed" mode of motion when they attach/detach to/from microtubules. Moreover, the two modes of motion are of greatly different speed. Since most existing tracking algorithms are specifically tailored to a particular type of motion, they can not be readily used.

5. Not every fluorescent point spread blob is a virus particle. They could also be created by virus-containing vesicles or caveolae (in case the caveolin-dependent pathway is used by the virus). Moreover, a single fluorescent blob might contain more than one virus particle and thus the point spread functions that one tracks can fuse or separate.

[2]requiring that neighboring particles should move in approximately the same way

In no way have we solved all of the above problems, but the tracking software described in this report tries to at least approximately consider them and points the way for future improvements.

This document is a technical report that describes the basic usage of the toolbox as well as its inner workings. Please also refer to the descriptive comment headers in every MATLAB file for further information on data formats and calling syntax. The software is currently in the stage of a "working prototype" and is certainly neither free of errors nor easy to use or in any sense complete in its functionality. It nevertheless serves its purpose, namely to extract trajectories of fluorescent virus particles from digital video frames. To successfully use the software one needs to have basic MATLAB skills and some understanding of what's going on in the algorithm in order to set the parameter values correctly. If the program does not yield the expected result, it is in most cases due to incorrect parameter settings. It may also be needed to make isolated changes in the source code of the program to integrate it into a specific environment or to satisfy specific needs.

2 Installation and basic usage

The distribution consists of the following files that can be downloaded from the CBL web page at `http://www.cbl.ethz.ch/Downloads/matlabtracker`:

- `tracker.m` (The main program)
- `detect_particles.m` (Feature detection to locate the particles in a frame image. Used by `tracker.m`)
- `link_trajectories.m` (Linking particle locations into trajectories over time. Used by `tracker.m`)
- `trackertest.m` (Utility to test and calibrate the tracker)
- `saturate.m` (Function to saturate the values in a vector.
  Used by `detect_particles.m`.)
- `normalize.m` (Function to normalize an image. Used by `tracker.m`.)
- `ll2matrix.m` (Function to convert a linked-list representation of trajectories to a matrix representation. Used by `mod.m`.)
- `mod.m` (Function to calculate the moments of displacement at different times and determine their scaling coefficients in time)

"Installation" simply means putting above files in a directory. In order to run the tracker, type `tracker(<filestub>, <ext>, <initial>, <final>)` at the MATLAB command prompt where `<filestub>` (string) is the constant stub of the input image file names (see section 3 for further information on input file format), `<ext>` (string) is the file extension (e.g. .tif) including the leading dot, `<initial>` (integer) is the number of the first frame image to be read and `<final>` (integer) the number of the last frame. The frame image files are exprected to be continuously numbered between `<initial>` and `<final>` with increments of one. The tracker loads all the image files into memory and performs some preprocessing (normalization) of the images. It then calls `detect_particles.m` for each frame to detect the positions of possible virus particles. The so obtained list of positions is passed to `link_trajectories.m` to link them into trajectories over time. Finally, the result is visualized in a figure window and stored to a file called `trackdata.mat`, which can later be read into matlab for subsequent processing.

3 Input data

`tracker.m` reads a series of image files corresponding to the frames of a digital video recording. The images are read as `<filestub>(<initial>...<final>)<ext>`. If you for example want to read TIFF images called `frame1.tif, frame2.tif, frame3.tif, ..., frame100.tif` that are in a subdirectory `movie1`, then you would set `filestub='movie1/frame'`, `initial=1`, `final=100` and `ext='.tif'`. The images can be in any format that is supported by the MATLAB Image processing toolbox (as of version 6.1: JPEG, TIFF, GIF, BMP, PNG, HDF, PCX, XWD, ICO, CUR). Notice that only the first channel of the image is read. For grayscale images, this is the only channel there is, but for color images it is the red channel. If you plan to use other channels of color images or a combination of channels (e.g. a KL transformation), you have to either preprocess the images in a image processing software or change the `imread` statement in `tracker.m` accordingly. Another limitation currently is that the program expects the image files to be continuously numbered between `initial` and `final` with increments of one and *without leading zeros in the number.* Again, you would only have to change the `imread` statement in `tracker.m` to alter that.

4 Parameters

All user-adjustable parameters are located in the first few lines of the file `tracker.m`. They are subdivided into "general parameters" and parameters for detection and linking. Table 1 lists the general parameters that are available and table 2 the detection and link parameters. To better understand the meaning and function of the detection and linking parameters, refer to section 9 of this document. Finally, see section 6 if you are operating in a distributed environment or on a multi-processor machine and want to speed the particle detection process up by distributing it over several processors or computers.

Parameter	Description	Default
`filestub`	Default file stub if none is specified as an argument.	''
`ext`	Default file extension if none is specified as an argument.	''
`para`	Run particle detection in parallel (1) or serial (0).	`0`
`parsys`	System used for parallel runs. One of 'MPI' (MATLAB Parallel Interface) or 'PM' (parMATLAB). See also section 6.	`'MPI'`
`viz`	Pop up figure windows for intermediate visualization during the process (1) or not (0).	`0`
`machines`	List of machine names (strings) to be used for MPI. Not used for parMATLAB.	{}

Table 1: General user parameters

5 Output

Upon completion of a successful run, `tracker.m` returns a cell list structure containing the detected trajectories. The same data structure is also stored to the binary

Parameter	Description	Default
`w`	Expected radius of particles in pixel (integer).	`3`
`cutoff`	Probability cutoff for non-particle discrimination.	`0.5`
`pth`	Intensity percentile in which to accept particles (in percent).	`1`
`L`	Maximum allowed displacement of a particle between frames (cutoff) in pixel.	`10`

Table 2: Detection and linking user parameters

file `trackdata.mat` that can later be read into MATLAB again using the `load` command. Each element of the cell list is a matrix containing all particle locations and intensity moments for a certain frame. If N_t particles were detected in frame image t, then `peaks{t}` is a $N_t \times 6$ matrix containing the following information:

`peaks{t}(p,1)`	x (row) position $\hat{x}_p$ of particle p in frame t.
`peaks{t}(p,2)`	y (column) position $\hat{y}_p$of particle p in frame t.
`peaks{t}(p,3)`	zero order intensity moment $m_0(p)$ of particle p in frame t.
`peaks{t}(p,4)`	second order intensity moment $m_2(p)$ of particle p in frame t.
`peaks{t}(p,5)`	unused
`peaks{t}(p,6)`	link list index pointing to the same (physical) particle in frame $t+1$, i.e. p in frame t and `peaks{t}(p,6)` in frame $t+1$ are subsequent points on the same trajectory. If p is the final point on a trajectory, this index is equal to -1.

This data structure can directly be used as input to `mod.m` or it can be converted into a matrix form using `ll2matrix.m`. The latter takes above data structure as an input and converts it to a cell list of matrices, each containing M_ℓ rows of (x_ℓ, y_ℓ)-vectors giving the positions along a particular trajectory ℓ of total length M_ℓ. Further information can be found in the comment headers of the corresponding program files.

6 Running in parallel

In order to speed up the process of particle detection in all the movie frame images, this step can be distributed over several processors or computers. Two different mechanisms of parallelization are currently implemented: parMATLAB and MPI. Both require (freely available) external toolboxes to be properly installed and configured in your MATLAB environment. It is not the scope of this report to describe the handling of those toolboxes, but nevertheless the basic concpets are outlined below.

6.1 Using MPI

MPI stands for "MATLAB Parallel Interface". It is a toolbox written by Einar Heiberg, that provides easy means of writing parallel programs, e.g. by providing a parallel version of `for` loops that automatically unroll across processors. It also mimics the Message Passing Interface (MPI) and makes its lower level calls available in MATLAB. It is based on the PMI toolbox (for the MPI command encapsulation) and the TCP/IP toolbox (for the communication). It makes use of low-level MEX files for performance reasons. One big advantage of this toolbox is that it does not need MATLAB daemons running on all processors. This however makes a machine list necessary in order to tell the toolbox which processors to use (cf. table 1). The toolbox and its documentation are freely available at `http://hem.passagen.se/einar_heiberg/index.html`.

6.2 Using parMATLAB

parMATLAB was developed by Lucio Andrade. It is a toolbox which distributes processes across MATLAB workers available over the intranet/internet. Every worker must be running a MATLAB daemon to be accessible. Communication is based on TCP/IP and makes use of low-level TCP/IP MEX files from the TCP/IP toolbox that first need to be compiled. It is very convenient to use since workers can be added or removed ”on the fly” by starting or stopping a MATLAB daemon on them. The master processor automatically detects any worker that is available and assigns work to it. parMATLAB offers many high-level parallelization commands to distribute the execution of a MATLAB function without explicitly having to deal with the message passing communication. The parMATLAB toolbox and its documentation can be freely downloaded from
`http://server1.cdsp.neu.edu/info/students/landrade/`.

7 Tests and validation

The distribution includes the utility program `trackertest.m` that can be used to test the tracker (and the parameter settings) as well as to measure the tracking quality (i.e. the RMS tracking error). `trackertest.m` creates series of artificial images of moving Gaussian intensity blobs and feeds them to `detect_particles.m` and `link_trajectories.m` for tracking of the blobs. The detected trajectories are then compared to the exact ones that created the motion in order to assess the tracking quality in terms of the length of the generated trajectories and the RMS deviation between the detected and the true trajectories. The type of motion as well as image and particle properties can be set in the parameters section at the beginning of the program. Table 3 lists the parameters and their meanings. Periodic boundary conditions are used in all directions, e.g. a particle that leaves the image at the right border will re-enter from the left.

Parameter	Description	Default
`type`	Type of motion. One of {`'linear'`,`'vortex'`,`'random'`}.	`linear`
`sigma`	Standard deviation (width) of the Gaussian blobs.	`2.0`
`speedx`	Speed of particles in row-wise direction in pixels per frame.	`0`
`speedy`	Speed of particles in column-wise direction in pixels per frame.	`5`
`idist`	Initial (average) inter-particle distance in pixel.	`20`
`siz`	Size of the created images (width=height) in pixel.	`512`
`noise`	Strength of pixel noise (inverse of signal to noise ratio).	`0.2`
`nframes`	Number of frames to be simulated.	`100`

Table 3: Tracker testing user parameters

Depending on the selected type of motion, the speed parameters have slightly different meanings and the initial particle configuration changes:

1. **linear**: Linear motion means motion of all particles along parallel straight lines. Initially, the particles are placed at the same column position but in different rows (with a separation of `idist`) as depicted by the left panel of figure 1. They then move along straight lines with the x- and y-components of the velocity vector given by v_x=`speedx` and v_y=`speedy`, respectively. Hence, the equations of motion for a given particle p are:

$$\begin{bmatrix} x_p(t) \\ y_p(t) \end{bmatrix} = \begin{bmatrix} x_0 + v_x t \\ y_0 + v_y t \end{bmatrix},$$

where $t = 0, \dots,$`nframes` is the current frame number and (x_0, y_0) is the particle's initial position. Notice that the effective speed of a particle is given by $\sqrt{v_x^2 + v_y^2}$. The right panel of figure 1 shows sample trajectories for `speedx=0, speedy=12`.

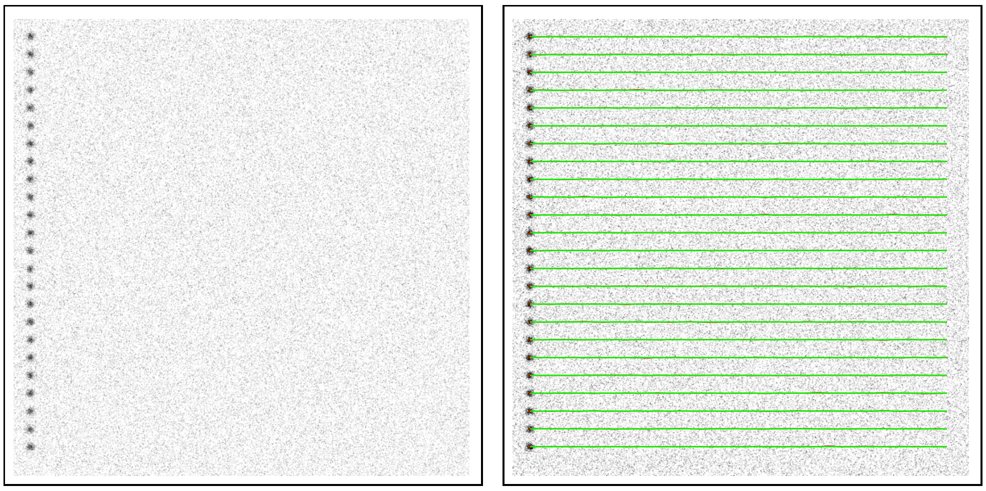


Figure 1: Linear motion with `speedx=0`, `speedy=12`, `sigma=2`, `idist=20`, `siz=512`, `noise=0.2`, `nframes=40`. Detected trajectories are shown in red, the true ones in green. (Image intensities inverted for printing purposes.)

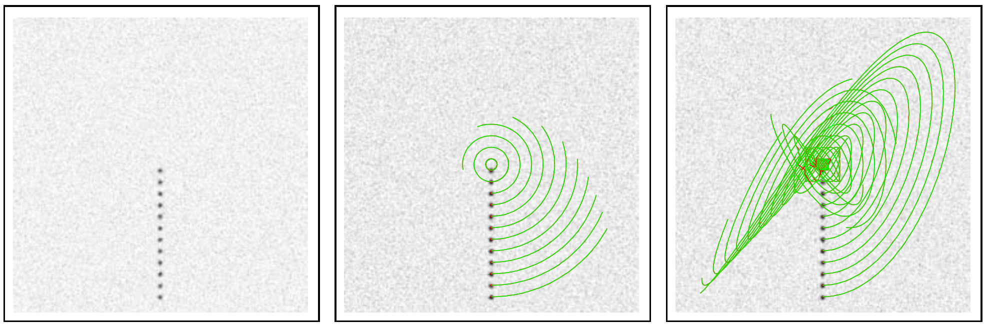
2. **vortex**: Vortical motion is governed by the equation:

$$\begin{bmatrix} x_p(t) \\ y_p(t) \end{bmatrix} = \begin{bmatrix} r \sin\left(t/(v_x r)\right) + m \\ r \cos\left(t/(v_y r)\right) + m \end{bmatrix},$$

where the radius r for a given particle is:

$$r = \sqrt{(m - x_0)^2 + (m - y_0)^2}$$

and m denotes the center of the image: m =`round(siz/2)`. Initially, all particles are assigned positions (x_0, y_0) in the same column of the image radially extending from the center of the image to the lower boundary with inter-particle separations of `idist` pixels (cf. left panel of figure 2). The observed motion depends on the settings of the speed parameters. For `speedx = speedy` the particles move in circles around the image center, each with a constant speed of $\sqrt{v_x^2 + v_y^2}$ (middle panel of figure 2). If the two speeds are different, the particles exhibit a more interesting motion along Lissajous curves (right panel of figure 2). If the two speeds are in an irrational ratio, these curves will never repeat themselves and the trajectories will eventually fill the whole image.



Initial particle positions　　Trajectories for $v_x = v_y = 7$　　Trajectories for $v_x = 5$, $v_y = 3$

Figure 2: Vortical motion for different `speedx` and `speedy`, `sigma=2`, `idist=20`, `siz=512`, `noise=0.2`, `nframes=50` (middle panel) or `nframes=200` (right panel). Detected trajectories are shown in red, the true ones in green. (Image intensities inverted for printing purposes.)

3. **random**: Random motion is used as a model for diffusing particles. The blobs are initially randomly scattered across the image with an *average* inter-particle distance of `idist` (left panel of figure 3). They then move according to:

$$\begin{bmatrix} x_p(t) \\ y_p(t) \end{bmatrix} = \begin{bmatrix} x_p(t-1) + \mathcal{N}(0, v_x^2 + v_y^2)\cos(\pi\mathcal{U}(0,1)) \\ y_p(t-1) + \mathcal{N}(0, v_x^2 + v_y^2)\sin(\pi\mathcal{U}(0,1)) \end{bmatrix}$$

for $t > 0$, where $\mathcal{N}(0, v)$ denotes normally distributed random numbers with expectation value 0 and variance v and $\mathcal{U}(0,1)$ are uniformly distributed random numbers between 0 and 1. This random process models free linear diffusion of the particles with an effective diffusion constant of $D = (v_x^2 + v_y^2)/4$ pixel^2/time unit. The right panel of figure 3 shows an example for $D = 6.25$ pixel^2/time unit.

8 Post-processing and analysis

Basic analysis of the acquired trajectories can be carried out by the utility function `mod.m`. It calculates the moments of displacement of the trajectories and determines their scaling coefficients over time. The input arguments to be supplied are:

1. The raw `trajectories` in linked list form as described in section 5.
2. A vector `delt` of integer frame interval values $[\Delta t_i]$ for which the moments are to be computed.
3. A vector `moments` of real-valued moment exponents $[\nu_j]$.

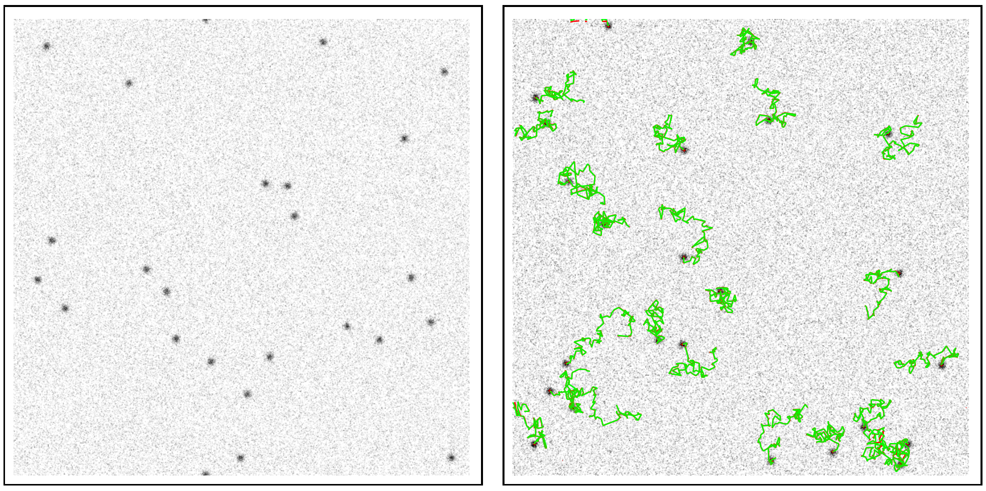


Figure 3: Random motion with `speedx=5`, `speedy=0`, `sigma=2`, `idist=20`, `siz=512`, `noise=0.2`, `nframes=100`. Detected trajectories are shown in red, the true ones in green. (Image intensities inverted for printing purposes.)

For each ν_j specified in the the argument `moments`, the function calculates the time course of the corresponding moment of the particle displacements:

$$\mu_\ell(\Delta t_i) = \frac{1}{M_\ell - \Delta t} \sum_{t=1}^{M_\ell - \Delta t} |\boldsymbol{x}_\ell(t + \Delta t) - \boldsymbol{x}_\ell(t)|^{\nu_j}$$

$$m(\Delta t_i) = \frac{1}{M} \sum_{\ell=1}^{M} \mu_\ell(\Delta t_i)\,,$$

where we have assumed that the data set consists of M trajectories. The total length (number of frames) of trajectory ℓ is denoted by M_ℓ. The vector $\boldsymbol{x}_\ell(t) = (x_\ell(t), y_\ell(t))$ denotes the position on trajectory ℓ in frame t and $|\cdot|$ is the 2-Norm. $\mu_\ell(\Delta t)$ is the ν_j-th moment of trajectory ℓ for the frame interval Δt and $m(\Delta t)$ is the average over all trajectories. Notice that this average is only meaningful if all trajectories in the data set perform the same kind of motion. It would be completely wrong if some particles would for example perform diffusive motion whereas others exhibit directed "ballistic" motion.

The scaling of the moments m vs. Δt for different ν_j are then considered by estimating the slope (doing a linear least squares fit) of $\log(m(\Delta t))$ vs. $\log(\Delta t)$. For the moment ν_j this slope is termed γ_{ν_j} and the vector $\gamma = \{\gamma_{\nu_j}, \forall \nu_j\}$ is returned. The function also displays a plot of the moment scaling spectrum γ_{ν_j} vs. ν_j, which gives important information about the type of motion observed. If the particles move in a purely diffusive way, this plot should show a straight line through the origin with slope 1/2. In the case of purely ballistic (i.e. steady and directed) motion, the plot should show a straight line trough the origin with slope 1. Every straight line trough the origin with a slope between 1/2 and 1 identifies superdiffusive motion and a slope between 0 and 1/2 characterizes subdiffusion. Lines that are not straight but have a kink somewhere indicate weakly self-similar diffusion [2], which is a non-linear process with a non-Gaussian kernel, even if the mean square displacement γ_2 is equal to 1.

9 Details of the algorithm

This section describes the basic algorithms that are implemented in the current version of the program. However, only the image processing part is described, split into the two parts "particle detection" and "trajectory linking". All post-processing, utility and data handling algorithms are not described.

Upon invocation, `tracker.m` loads all frame images into memory. It then determines the *global*[3] minimum and maximum of all intensity values occurring in the movie. These global extrema are used in a pre-processing step to normalize the intensities in all images such that they all are between 0 and 1. Using global extrema instead of local (per frame) ones preserves intensity variations across frames, which is an important source of information in the linking step. After this pre-processing, `detect_particles.m` is called for every frame image to get a complete list of particle positions which is finally handed to `link_trajectories.m`.

9.1 Particle detection

The feature (particle) detection consists of 4 steps:

1. Image restoration
2. Estimating the particle locations
3. Refining the particle locations
4. Non-particle discrimination

The implemented algorithm is based on the one proposed by Crocker and Grier [1] for the detection of gold colloids in micrographs. In the following, the frame image at frame t is represented as a matrix $A^t(x, y)$ of floating point intensity values between 0 and 1. The integer coordinate $x = 1, \dots, N_x$ is the pixel row index (total height of the image is N_x) and $y = 1, \dots, N_y$ the pixel column index. All neighborhood pixel operations described below implicitly imply truncation boundary conditions, i.e. only existing pixels are involved and the images are not padded.

The **image restoration** step tries to correct imperfections in the images. There are two different effects accounted for: (1) long-wavelength modulations of the background intensity due to non-uniform sensitivity among the camera pixels or uneven illumination and (2) Discretization noise from the digital camera and the frame grabber. The former is quite easy to correct for since no information is lost and the particles are small compared to background variations and thus well separated in frequency. The background is reasonably well removed by a boxcar average over a square region of extent $2w + 1$ pixel:

$$A_w^t(x, y) = \frac{1}{(2w+1)^2} \sum_{i=-w}^{w} \sum_{j=-w}^{w} A^t(x+i, y+j)\,,$$

where the user-parameter w (cf. table 2) is an integer larger than a single particle's apparent radius but smaller than the smallest inter-particle separation. Camera noise is modeled as purely random and uniform with a correlation length of $\lambda_n = 1$ pixel. The filter then consists of a convolution of the image A^t with a Gaussian surface of half width λ_n:

$$A_{\lambda_n}^t(x, y) = \frac{1}{B} \sum_{i=-w}^{w} \sum_{j=-w}^{w} A^t(x+i, y+j) \exp\left(-\frac{i^2+j^2}{4\lambda_n^2}\right),$$

[3] i.e. across all the frames of the movie rather than within each frame separately

with normalization $B = \left[\sum_{i=-w}^{w} \exp\left(-\left(i^2/(4\lambda_n^2)\right)\right)\right]^2$. Since both equations can be written as convolutions of the image with kernels of support $2w+1$, the steps are combined and the final image restoration simply amounts to a convolution of the original image with the kernel

$$K(i,j) = \frac{1}{K_0}\left[\frac{1}{B}\exp\left(-\frac{i^2+j^2}{4\lambda_n^2}\right) - \frac{1}{(2w+1)^2}\right].$$

The normalization constant

$$K_0 = \frac{1}{B}\left[\sum_{i=-w}^{w}\exp\left(-\frac{i^2}{2\lambda_n^2}\right)\right]^2 - \frac{B}{(2w+1)^2}$$

allows comparison among images filtered with different values of w. To do the convolution, the image is temporarily padded by repeating the first and last row and column w-fold outwards[4] in order to get the correct convolution result at the boundary (of thickness w) of the original image. Negative pixel values generated by the convolution are reset to 0, since they are an artifact of the assumption of symmetric Gaussian camera noise (which is of course not true for the zero intensity level as only the positive part of the noise exists there) and in principle, one-sided kernels would have to be used for the boundary intensity levels 0 and 1. For kernel sizes above a certain limit, above convolution can efficiently be performed using fast Fourier transforms. For w smaller than about 6 (depending on the particular microprocessor architecture), the direct evaluation of the convolution sum is faster than the FFT method, since it involves less overhead. Figure 4 shows an example image from work done on Polyoma virus by Dr. Alicia Smith[5] both before and after applying above filter for image restoration. The improvement in image quality can clearly be seen and the background is effectively filtered out without significantly blurring the particle spots.

Estimating the particle locations is done by locating local intensity maxima in the filtered image. Hereby, a pixel is taken as the approximate location of a particle if no other pixel within a distance of w is brighter and if its intensity is in the upper p-th percentile of intensity values of the entire image. The regional maximum selection is implemented as a grayscale dilation[6] followed by the selection of all pixels that had the same value before and after the dilation. If such a pixel is in the upper p-th (parameter `pth` in table 2) percentile of intensity values, it is taken as the candidate location of a particle. The percentile criterium is needed to prevent the algorithm from selecting background pixels (that have intensity 0 both before and after the dilation) as particle locations.

The regional maximum selection of particle centers suffers from two deficiencies: (1) it is unable to reject noise, which leads to errors in the location estimate of particles and (2) it will include false identifications such as random bright particles in the background of the image or fluorescent vesicles or organelles. This makes both a refinement of the detected particle locations and a subsequent non-particle discrimination necessary.

Refining the particle locations will reduce the standard deviation of the position measurement to less than 0.1 pixel [1] even with moderate signal to noise. Other information gathered in the process can furthermore be reused later to reject spurious identifications. The assumption is that the found regional maximum of a

[4]but the convolution is only applied to the original pixels and not the added ones.

[5]Institute of Biochemistry, ETH Zürich.

[6]A grayscale dilation is an elementary morphological operation which sets the value of a pixel $A(x,y)$ to the maximum value found within a distance of w.

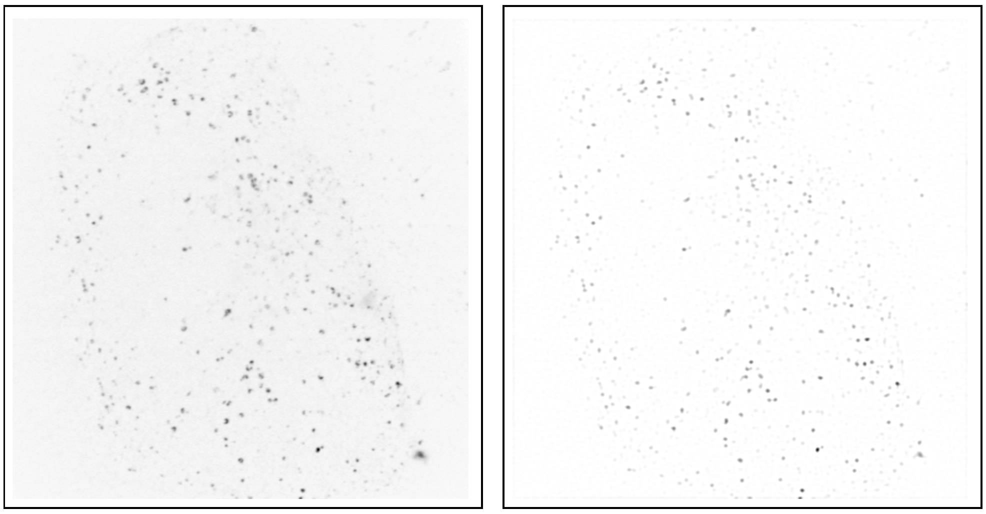


Figure 4: Confocal image of fluorescently labeled Polyoma virus on PTK2 cells 1 hour post infection (image courtesy of Dr. Alicia Smith) before (left panel) and after (right panel) the convolution filtering with `w=3`. (Image intensities inverted for printing purposes.)

particle p at $(\hat{x}_p, \hat{y}_p)$ is near the particle's geometric center (x_p, y_p). An approximation of the offset is given by the distance to the brightness-weighted centroid in the filtered image A^t:

$$\begin{bmatrix} \varepsilon_x(p) \\ \varepsilon_y(p) \end{bmatrix} = \frac{1}{m_0(p)} \sum_{i^2+j^2 \leqslant w^2} \begin{bmatrix} i \\ j \end{bmatrix} A^t(\hat{x}_p + i, \hat{y}_p + j)\,.$$

The normalization factor $m_0(p)$ is the sum of all pixel values over a particle p, i.e. its intensity moment of order 0:

$$m_0(p) = \sum_{i^2+j^2 \leqslant w^2} A^t(\hat{x}_p + i, \hat{y}_p + j)\,.$$

The location estimate is now refined as: $(\tilde{x}_p, \tilde{y}_p) = (\hat{x}_p + \varepsilon_x(p),\, \hat{y}_p + \varepsilon_y(p))$. If either $|\varepsilon_x(p)|$ or $|\varepsilon_y(p)|$ is larger than 0.5, the candidate location $(\hat{x}_p, \hat{y}_p)$ is accordingly moved by 1 pixel and the refinement recalculated.

The **non-particle discrimination** finally should reject false identifications such as autofluorescence signals, dust or fluorescent vesicles. The implemented algorithm is based on the intensity moments of orders 0 and 2. The 0^{th} order moment of each particle p has already been calculated in the previous step as:

$$m_0(p) = \sum_{i^2+j^2 \leqslant w^2} A^t(\hat{x}_p + i, \hat{y}_p + j)\,.$$

The second order intensity moment is calculated as:

$$m_2(p) = \frac{1}{m_0(p)} \sum_{i^2+j^2 \leqslant w^2} \left(i^2 + j^2\right) A^t(\hat{x}_p + i, \hat{y}_p + j)\,,$$

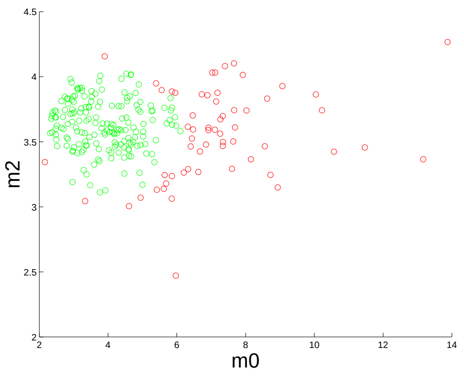
where $(\hat{x}_p,\, \hat{y}_p)$ are the coordinates of the refined location of the centroid of particle p. Since every virus particle gives raise to an apparent point spread function, the observations will fall into a narrow cluster in the (m_0, m_2)-plane. Larger and brighter structures such as vesicles or virus accumulations will have a larger m_0 and fall outside of the main cluster. Spurious identifications such as fluorescent parts of organelles or secondary vesicles are larger and/or more dim and thus have a different m_2 moment which causes them to fall outside of the cluster, too. The basic assumption in this is that the majority of the identifications correspond to correct virus particles such that the majority of the points in the (m_0, m_2)-plane forms a dense cluster. This cluster is identified by having each particle p "carry" a 2D Gaussian:

$$P_p = \frac{1}{2\pi\sigma_0\sigma_2} \exp\left(-\frac{(m_0 - m_0(p))^2}{2\sigma_0} - \frac{(m_2 - m_2(p))^2}{2\sigma_2}\right)$$

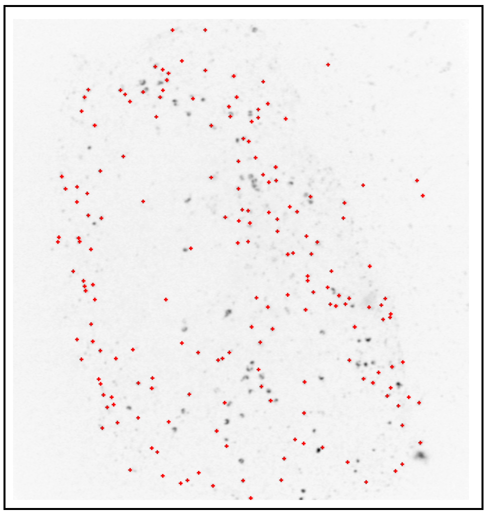
with standard deviations $\sigma_0 = 0.1$ in the m_0 direction and $\sigma_2 = 0.1$ in the m_2 direction. The contributions of all other particles $q \neq p$ are then summed for each particle p at its location:

$$S_p = \sum_{q \neq p} P_q \, .$$

Every particle identification having such a score above a certain user-provided threshold `cutoff` (cf. table 2) is considered a true virus particle, all others are discarded. Figure 5 illustrates this using the same sample data as figure 4. The final particle detections are shown in the right panel and marked with a red plus. It can be seen that structures probably corresponding to secondary endocytic vesicles, autofluorescence or noise seem to be mostly rejected.



Cluster in (m_0, m_2)-plane



Final particle detections

Figure 5: Left panel: Clustering in the plane of intensity moments for the sample data of figure 4 with `cutoff=0.9`, `pth=1`, $\sigma_0 = \sigma_2 = 0.1$. Green circles mark points that are kept as real particle detections, red ones are rejected. Right panel: Final particle detections marked by red crosses. (Image intensity inverted for printing purposes.)

9.2 Trajectory linking

Having done above particle detection algorithm for each frame image A^t leaves us with a set of T (total umber of frames) matrices $C^t \in \mathbb{R}^{N_t \times 2}$ with rows $[\tilde{x}_p, \tilde{y}_p]_{p=1}^{N_t}$ where the total number of particles detected in frame t is denoted by N_t. The task is now to identify observations of the same physical particle in subsequent frames in order to link the position readings $\{C^t\}_{t=1}^{T}$ into trajectories over time. The implemented algorithm approximately solves this problem using a modification of the *transportation algorithm*, taking a graph theory approach as described in `http://www.damtp.cam.ac.uk/user/fdl/people/sd/digimage/document/track2d/matching.htm`. The basic idea is to determine a set of associations between two sets of particle locations such that the set of associations is optimal in the sense that it minimizes a linear cost functional. One set is the set $\mathcal{P}$ of particles p_i, $i = 1, \ldots, N_t$ in frame t, the other is the set $\mathcal{Q}$ of particles q_j, $j = 1, \ldots, N_{t+1}$ in frame $t + 1$. We now define an association matrix G with element $G(i, j) = g_{ij}$ equal to 1 if p_i in frame t and q_j in frame $t + 1$ are produced by the same physical particle, 0 otherwise.

For now, we assume that there is always exactly one physical particle producing a single particle position reading. This obviously is a limiting assumption since (as mentioned in the introduction) virus particles can coalesce in vesicles or come so close that they are indistinguishable by the microscope, giving raise to one single point spread blob. Moreover the particle detection algorithm is not free of errors and nearby blobs are sometimes detected as only one. This limitation of the algorithm is certainly something to be overcome in future versions.

In order to allow the number of particles to vary between frames, i.e. $N_t \neq N_{t+1}$, the association matrix is augmented with both a row g_{0j} and a column g_{i0} for *dummy particles* at times t and $t + 1$. Linking a particle to the dummy means that it disappeared from the movie between frames t and $t+1$ and linking the dummy to a particle means that it newly appeared. With this in place and above assumption, the following topology constraint on the matrix G can be formulated:

Every row $i > 0$ of G and every column $j > 0$ of G must contain exactly one entry of value 1, all others are zero. Row 0 and column 0 are allowed to contain more than one 1.

In order to find an optimal set of links g_{ij}, we need to define the functional to be minimized. The only restriction in the definition of such a functional is that is needs to be linear in the association variables g_{ij} and may thus be written as:

$$\Phi = \sum_{i=0}^{N_t} \sum_{j=0}^{N_{t+1}} \phi_{ij} g_{ij} ,$$

where ϕ_{ij} represents the cost of associating particle p_i in frame t with particle q_j in frame $t + 1$. The definition of ϕ typically includes the particle positions, characteristics or (if available) temporal and spatial knowledge about the physics of the process. In our case, we use the quadratic distance between p_i, $i > 0$ and q_j, $j > 0$ as well as the quadratic differences in the intensity moments of order 0 and 2, thus:

$$\phi_{ij} = \left(\tilde{x}_{p_i} - \tilde{x}_{q_j}\right)^2 + \left(\tilde{y}_{p_i} - \tilde{y}_{q_j}\right)^2 + \left(m_0(p_i) - m_0(q_j)\right)^2 + \left(m_2(p_i) - m_2(q_j)\right)^2$$

for $i, j > 0$. The cost of linking a particle to one of the dummy particles $i = 0$ and $j = 0$ is set equal to the square of a parameter L (cf. table 2): $\phi_{0j} = L^2$, $j > 0$, and $\phi_{i0} = L^2$, $i > 0$. The special case of linking a dummy to a dummy, i.e. $i = j = 0$

is set to $\phi_{00} = 0$. This effectively places a limit to the allowed cost for particle associations since no association of cost larger than L^2 will happen between regular particles because the dummy association would be more favorable. The parameter L can therefore be interpreted as the maximum distance a particle is allowed to travel between frames (given its intensity moments remain constant). All costs $\{\phi_{ij} : \phi_{ij} > L^2\}$ are set to ∞ and the corresponding g_{ij} will never be considered in the following, since these matches will never occur.

Initialization. The association matrix G is initialized by assigning each particle in frame t its "nearest neighbor" (using the distance measure ϕ) in frame $t+1$ that is not already assigned to some other particle. This means that for every given $i = I$, $j = J$ is chosen such that ϕ_{IJ} is the minimum of all ϕ_{Ij} for which no other g_{iJ} is already set to one. This g_{IJ} is then set to one. If no such minimum is found, the particle is linked to the dummy, i.e. g_{I0} is set to one. After having done this for all the particles p_i, every J for which no g_{iJ} is set is determined and the corresponding g_{0J} is set to one. This initialization generates a matrix G that fulfills above-mentioned topology constraint. Frequently this initial solution will already be very close to optimal since only few conflicts (i.e. the association that would have the lowest cost is already blocked by another one) will occur in practice. Nevertheless, the association matrix is iteratively optimized in the following.

Optimization. For each iteration, we scan trough all g_{ij} (including the dummy particles) that are equal to zero *and* have finite associated cost ϕ_{ij}, and determine the *reduced cost* of introducing that association into the matrix. To see how the reduced cost is calculated from the elementary costs ϕ for i, $j > 0$, consider a zero association g_{IJ}, I, $J > 0$. Suppose that $g_{IL} = 1$ and $g_{KJ} = 1$ (since every row and column must contain a 1 according to the topology constraint). Now if g_{IJ} was to be set to one, then g_{IL} and g_{KJ} must turn zero, otherwise particles p_I and q_J would be in two places at once. Further, as particle images $i = K$ and $j = L$ must be related to some physical particle, it is necessary to set $g_{KL} = 1$. The *reduced cost of setting g_{IJ}, I, $J > 0$, to one* thus is:

$$z_{IJ} = \phi_{IJ} - \phi_{IL} - \phi_{KJ} + \phi_{KL} \qquad I,\, J > 0\,.$$

If the reduced cost z_{IJ} is negative, then introducing the association g_{IJ} into the solution is favorable, i.e. will decrease the cost functional Φ. In the case where $i = 0$, i.e. the zero association is at g_{0J} for some $J > 0$, only the 1 in the same column at g_{KJ}, $K > 0$, is turned into a 0 and the dummy entry g_{K0} is set to 1. The reduced cost thus is:

$$z_{0J} = \phi_{0J} - \phi_{KJ} + \phi_{K0} \qquad J,\, K > 0,\, L = 0\,.$$

For $j = 0$ we similarly find:

$$z_{I0} = \phi_{I0} - \phi_{IL} + \phi_{0L} \qquad I,\, L > 0,\, K = 0\,.$$

setting g_{I0}, $I > 0$ to 1, turning g_{IL}, $L > 0$, from 1 to 0 and setting the dummy g_{0L} to 1 as well. The special case $I = J = 0$ is set to $z_{00} = 0$. After calculating the reduced costs $\forall\{(i,j) : g_{ij} = 0 \wedge \phi_{ij} < \infty\}$, the g_{IJ} which had the most negative associated reduced cost $z_{IJ} = \min_{i,j} z_{ij}$ is set to 1, the corresponding g_{IL} (if $i \neq 0$) and g_{KJ} (if $j \neq 0$) to zero and g_{KL} to one, in order to preserve the topology of G. Then, all the reduced costs are recalculated and the whole iteration is repeated until $z_{ij} \geqslant 0$, $\forall\,(i,j)$, which means that the optimal solution (with cutoff L) has been found.

Repeating above procedure for every pair of frames $(t,\, t+1)$ in the movie leads to an optimal (in the sense of the chosen cost functional Φ) linking of the particle observations into trajectories over time.

References

[1] J. C. Crocker and D. G. Grier. Methods of digital video microspopy for colloidal studies. *J. Colloid Interf. Sci.*, 179:298–310, 1996.

[2] R. Ferrari, A. J. Manfroi, and W. R. Young. Strongly and weakly self-similar diffusion. *Physica D*, 154:111–137, 2001.