Methods for particle tracking in zebrafish

Yngve Mardal Moe

November 2021

1 Introduction

This document describes the workflow for tracking particles in zebrafish embryo. The workflow is separated into three parts.

- Preprocessing and particle tracking
- Estimating vasculature geometry
- Estimating velocities and pressures

We disregarded the results for all files where the vessel background image had different shape compared to the particle videos.

2 Preprocessing and particle tracking

Tracking particles is done using `trackpy`, a Python library for PTV. TrackPy uses the method described in [2], which roughly speaking does as follows:

1. Preprocess each frame using a band-pass filter. This removes high-frequency noise and low-frequency sensor differences. The high-pass part of the band-pass filter is decided based on the particles' estimated size
2. Find possible particles. This will find a large amount of false positives that needs to be filtered away.
3. Filter away false positive particles. We require that particles are separated by at least 6 pixels and that the particles have total mass (i.e. total integrated brightness) ≥ 50.

4. Link particles between frames by searching the next frame in a 16 pixel radius around the particle's position in the current frame. Our settings allows particles to be present for only every second frame.

5. Exclude particles that are not present in at least two frames.

Step 1-3 are performed by the `trackpy.batch` function, step 4 is performed by the `trackpy.link` function and step 5 is performed by the `trackpy.filter_stubs` function. We noticed, however, that this default preprocessing is insufficient. We therefore perform the following preprocessing instead

1. For each pixel, compute its average value across each frames. This gives us a background-signal estimate

2. For each pixel, subtract the background signal and set all negative pixels equal to 0

3. Perform a morphological (greyscale) opening with a 3×3 structuring element

4. Perform a morphological (greyscale) closing with a 5×5 structuring element

5. Modify the dynamic range to be between 2 and 50 (clipping pixel values between 2 and 50)

6. Transform each pixel value using the transform $T(x) = 255(x-2)/(50-2)$

By disabling the default preprocessing in `trackpy.batch` and using the above preprocessing instead, we get a better estimate of the particle positions.

To track particles, with this pipeline, use the `scripts/track_particles.py` script.

3 Estimating vasculature geometry

The next step is to estimate the vasculature geometry. Specifically its shape, centerline and radius. To estimate the vasculature shape and centerline, we used a manual segmentation procedure (with the `scripts/roi_generator.py` script).

3.1 Estimating shape and centerline

The manual segmentation process worked as follows:

1. Add first point on vessel boundary
2. Add new point on vessel boundary. The outline will be linearly interpolated between these this point and the previous point.
3. Repeat the above step until the full outline is drawn. Once finished, the first and last point will be connected.

After creating a segmentation mask (or region of interest, ROI), we need to parametrise the centerline. To accomplish this, we use the following process

1. Skeletonise the ROI (Lee's algorithm [3])
2. Find the index of all skeleton-pixels.
3. Compute the 2-nearest-neighbours (2NN) graph using the nonzero-pixel coordinates.
4. Find endpoints of the skeletonised ROI. This is done by convolving the skeletonised ROI with a 3×3-kernel consisting of only ones. The endpoints will then be the pixels with value equal to 2.
5. Find the shortest path in the 2NN-graph between the two edge pixels. This will give an ordered list of coordinates, which we can interpolate between to compute the centerline. We use nearest-neighbour interpolation for this, since that makes future steps easier to implement, but higher-order splines are also possible.
6. Estimate the direction of the centerline at the two ends of the centerline. We use a finite-difference approximation, but "looking" two pixels back instead of one to reduce the chance of a $45°$-angle.
7. Cut the ROI so the centerline-endpoints touch the boundary of the ROI.

There are two important notes to be aware of when creating the ROI. Firstly, the skeletonisation must return a single line with no bifurcations, otherwise, the centerline parametrisation will not work correctly. As a consequence, the ROI must also be drawn in with no birfurcations. Also you should not use a straight line when you "cut" the ROI. Instead, we should have a "arrow"-like shape that points out of the ROI (see Figure 1). This is important for the skeletonisation procedure. Otherwise, we may end up with a skeleton with slight bifurcations towards the corners of the ROI.

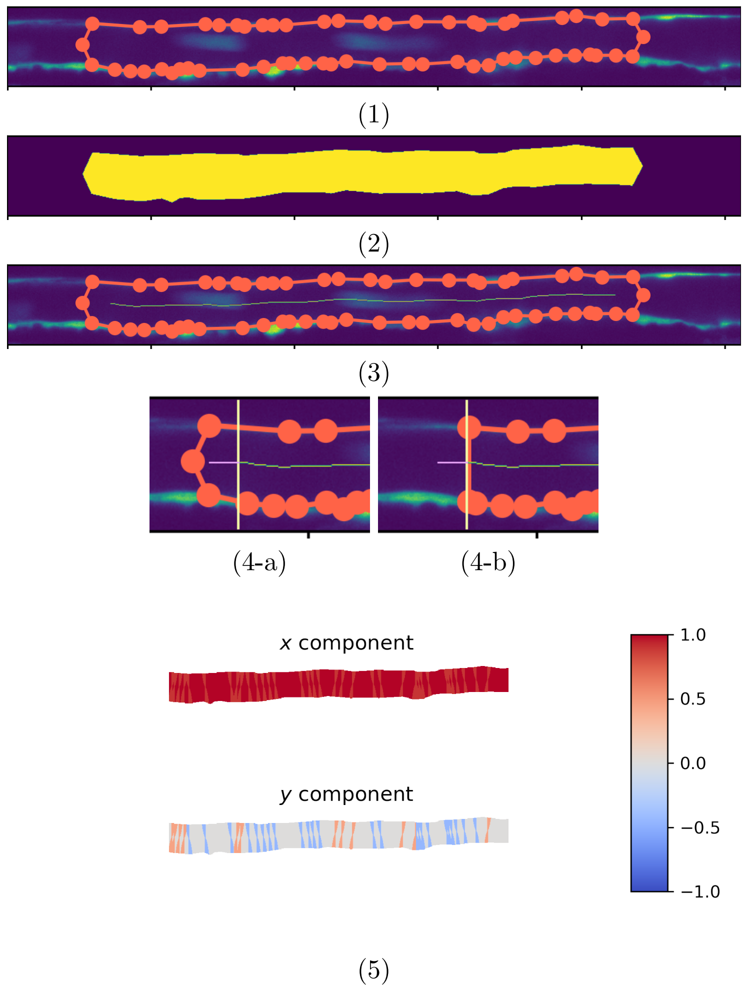

Figure 1: From top to bottom: (1) The manually marked ROI, each dot represents a vertex manually added by the user. (2) The polygonal mask generated from the outline. (3) The skeleton of the polygonal mask, with the ROI superimposed. (4-a) The polygonal mask before clipping, in pink, the estimated direction, in yellow, the centerline normal, used to clip the ROI. (4-b) The polygonal mask after clipping, in pink, the estimated direction, in yellow, the centerline normal, used to clip the ROI. (5) The directional components of the nearest centerline pixel.

3.2 Estimating centerline-distances, and vessel radius

To estimate the distance to the centerline, we transform the skeletonised ROI with the euclidean distance transform and (bi-)linearly interpolate subpixel differences in the particle positions. To estimate the radius of the blood vessel, we use the maximum distance to the centerline for all particles within the ROI.

4 Estimating velocities

To estimate the velocities, we need three things

1. The particle tracks
2. The spatial pixel-dimensions
3. The time between each frame

The particle tracks were already estimated with the `scripts/track_particles.py`-script. However, that script didn't filter tracks by their length. Here, we filter those tracks further by removing all tracks where the particle was present for less than 5 frames. We also remove all tracks outside the ROI.

To compute the spatial pixel-dimensions and the time between each frame, we use the image metadata, obtained with the `confocal_microscopy.files.ims.load_ims_metadata`-function. We then use the following code to get the image size, pixel size, and timestep. The output of this code is compared with many of the `legend.docx` files provided by Federico, and these numbers always coincided.

```
## Get the image metadata
image_size = ims.find_physical_image_size(metadata)[1:]
image_shape = [
    int(metadata["CustomData"]["Height"]),
    int(metadata["CustomData"]["Width"])
]
pixel_size = np.round(np.array(image_size) / image_shape, 3)

timestamps = [
    datetime.fromisoformat(metadata["TimeInfo"][f"TimePoint{i+1}"])
    for i in range(7000)
]
timestamps = [t - timestamps[0] for t in timestamps]
timestamps = [(t.seconds + t.microseconds*1e-6) for t in timestamps]
```

```
    timestamps = np.linspace(0, timestamps[-1], len(timestamps))

    timestep = timestamps[-1]/len(timestamps)
```

Based on this, we could transform the particle velocities from pixels per frame to µm/s.

4.1 Validating the results

To validate the results from the particle tracking, we manually tracked in a variety of of fishes and vessels. The automatic velocity estimates were generally satisfying with a high sensitivity[1] (most manually tracked particles were found), and a low-medium positive predictive value[2] (the automatic tracking found approximately twice the number of tracks compared to the manual tracking). Some of the tracks found by the automatic algorithm and not manually were false positives and others were actual particles not found during the manual tracking.

To see the results from the manual tracking, see the `scripts/Early exploratory analysis.ipynb`.

5 Estimating pressures

There are several ways to estimate the pressure. If we first assume that we know the viscosity (more about that later), then we can estimate the pressure by

1. Using Poiseuille's law and estimating the pressure for each particle independently, and then computing the average pressure gradient
2. Assume a velocity profile on the form $v(r) = br^2 - a$ and compute the pressure gradient analytically

With option one, we estimate the pressure-gradient that drives the motion of the i-th particle, $|\nabla p_i|$, with the following formula

$$|\nabla p_i| = 3v_i\mu/(R - r_i), \tag{1}$$

[1]Also known as recall
[2]Also known as precision

where v_i and r_i is the velocity and distance to centerline for the i-th particle, R is the radius of the vessel and μ is the blood viscosity. To compute the total pressure gradient, $|\nabla p|$, we compute the average of $|\nabla p_i|$.

If we instead assume a monomial+intercept velocity profile, then we estimate the total pressure gradient, $|\nabla p|$, by

$$|\nabla p| = 4a\mu/R^2, \tag{2}$$

where a is the amplitude of the velocity profile. To estimate the parameters of the velocity profile, we use the fact that $v(R) = 0$ to remove one degree of freedom (the b in $v(r) = br^2 + a$). Then, we use Brent's root-finding method [1, 6] to estimate the a that minimise the squared error $\sum_i (v(r_i) - v_i)^2$.

To estimate the velocities and pressures for a single blood vessels, run the `scripts/Summary analysis.ipynb`-notebook. To compute for all blood vessels, run the `scripts/compute_summaries.py` function.

5.1 Estimating the viscosity

There are two ways to estimate the viscosity, either with the method of Lee et al. [4], who measured the viscosity as a function of tube hematocrit (ht) (fraction red blood cells in the blood) in a rectangular channel of size $60\mu\text{m} \times 240\mu\text{m}$. They found that the blood was Newtonian in so large channels, and fitted a quadratic model to the viscosity for $0 \leq ht \lessapprox 0.35$. Moreover, Lee et al. found that zebrafish blood plasma ($ht = 0$) has approximately 1.5 the viscosity of water.

The other way of estimating the viscosity of zebrafish blood is with the method of Pries et al. [5], who combined many estimates of human blood viscosity and found that the viscosity varies greatly with the vessel radius and discharge hematocrit (fraction of red blood cells in blood that is released through an open blood vessel). They fitted a complicated heuristic model based on a variety of properties of human blood. One of which was that human blood plasma has the same viscosity as water. This model is therefore not fully compatible with the findings of [4].

To conclude, we see that there is no accurate measurement of zebrafish blood viscosity. The measurements by Lee et al. does not account for small vessels, where the red blood cells interact with the vessel walls, whereas the model by Pries et al. is based on human blood, which has different properties compared to the blood of zebrafish.

References

[1] Richard P Brent. *Algorithms for Minimization without Derivatives, chap. 4.* 1973.

[2] John C Crocker and David G Grier. “Methods of digital video microscopy for colloidal studies”. In: *Journal of colloid and interface science* 179.1 (1996), pp. 298–310.

[3] Ta-Chih Lee, Rangasami L Kashyap, and Chong-Nam Chu. “Building skeleton models via 3-D medial surface axis thinning algorithms”. In: *CVGIP: Graphical Models and Image Processing* 56.6 (1994), pp. 462–478.

[4] Juhyun Lee et al. “A rapid capillary-pressure driven micro-channel to demonstrate newtonian fluid behavior of zebrafish blood at high shear rates”. In: *Scientific reports* 7.1 (2017), pp. 1–8.

[5] Axel R Pries, D Neuhaus, and P Gaehtgens. “Blood viscosity in tube flow: dependence on diameter and hematocrit”. In: *American Journal of Physiology-Heart and Circulatory Physiology* 263.6 (1992), H1770–H1778.

[6] Pauli Virtanen et al. “SciPy 1.0: fundamental algorithms for scientific computing in Python”. In: *Nature methods* 17.3 (2020), pp. 261–272.