flowcatchR: A framework for tracking and analyzing flowing blood cells in time lapse microscopy images

Federico Marini[1], Johanna Mazur, Harald Binder

Institute of Medical Biostatistics, Epidemiology and Informatics (IMBEI),
University Medical Center - Mainz (Germany)

[1]marinif@uni-mainz.de

Edited: August 20, 2015; Compiled: October 17, 2016

Abstract

flowcatchR is a set of tools to analyze in vivo microscopy imaging data, focused on tracking flowing blood cells. *flowcatchR* guides throughout all the steps of bioimage processing, from segmentation to calculation of features, filtering out particles not of interest, providing also a set of utilities to help checking the quality of the performed operations. The main novel contribution investigates the issue of tracking flowing cells such as the ones in blood vessels, to categorize the particles in flowing, rolling, and adherent by providing a comprehensive analysis of the identified trajectories. The extracted information is then applied in the study of phenomena such as hemostasis and thrombosis development. We expect this package to be potentially applied to a variety of assays, covering a wide range of applications founded on time-lapse microscopy.

Contents

1 Introduction

This document offers an introduction and overview of the *R*/*Bioconductor* [1, 2] package *flowcatchR*, which provides a flexible and comprehensive set of tools to detect and track flowing blood cells in time-lapse microscopy.

1.1 Why flowcatchR?

flowcatchR builds upon functionalities provided by the *EBImage* package [3], and extends them in order to analyze time-lapse microscopy images. Here we list some of the unique characteristics of the datasets *flowcatchR* is designed for:

- The images come from intravital microscopy experiments. This means that the Signal-to-Noise Ratio (SNR) is not optimal, and, very importantly, there are potential major movements of the living specimen that can be confounded with the true movements of the particles of interest [4]
- Cells are densely distributed on the images, with particles that can enter and leave the field of view
- The acquisition frame rate is a compromise between allowing the fluorescent cells to be detected and detecting the movements properly
- Cells can flow, temporarily adhere to the endothelial layer and/or be permanently adherent. Therefore, all movement modalities should be detected correctly throughout the entire acquisition. Cells can also cluster together and form (temporary) conglomerates

Essential features *flowcatchR* delivers to the user are:

- A simple and flexible, yet complete framework to analyze flowing blood cells (and more generally time-lapse) image sets, with a system of S4 classes such as *Frames*, *ParticleSet*, and *TrajectorySet* constituting the backbone of the procedures
- Techniques for aiding the detection of objects in the segmentation step
- An algorithm for tracking the particles, adapted and improved from the proposal of Sbalzarini and Koumoutsakos (2005) [5], that reflects the directional aspect of the motion
- A wide set of functions inspecting the kinematic properties of the identified trajectories [6, 7], providing publication-ready summary statistics and visualization tools to help classifying identified objects

This guide includes a brief overview of the entire processing flow, from importing the raw images to the analysis of kinematic parameters derived from the identified trajectories. An example dataset will be used to illustrate the available features, in order to track blood platelets in consecutive frames derived from an

intravital microscopy acquisition (also available in the package). All steps will be dissected to explore available parameters and options.

To install the package *flowcatchR*, please start a current version of R and type

1.2 Purpose of this document

This vignette includes a brief overview of the entire processing flow, from importing the raw images to the analysis of kinematic parameters derived from the identified trajectories. An example dataset will be used to illustrate the available features, in order to track blood platelets in consecutive frames derived from an intravital microscopy acquisition (also available in the package). All steps will be dissected to explore available parameters and options.

2 Getting started

2.1 Installation

flowcatchR is an R package distributed as part of the Bioconductor project. To install *flowcatchR*, please start R and type:

```
source("http://bioconductor.org/biocLite.R")
biocLite("flowcatchR")
```

In case you might prefer to install the latest development version, this can be done with these two lines below:

```
install.packages("devtools") # if needed
devtools::install_github("federicomarini/flowcatchR")
```

Installation issues should be reported to the Bioconductor support site (http://support.bioconductor.org/).

2.2 Getting help

The *flowcatchR* package was tested on a variety of datasets provided from cooperation partners, yet it may require some extra tuning or bug fixes. For these issues, please contact the maintainer - if required with a copy of the error messages, the output of `sessionInfo` function:

```
maintainer("flowcatchR")
```

```
## [1] "Federico Marini <marinif@uni-mainz.de>"
```

Despite our best efforts to test and develop the package further, additional functions or interesting suggestions might come from the specific scenarios that the package users might be facing. Improvements of existing functions or development of new ones are always most welcome! We also encourage to fork the GitHub repository of the package (https://github.com/federicomarini/flowcatchR), develop and test the new feature(s), and finally generate a pull request to integrate it to the original repository.

2.3 Citing flowcatchR

The work underlying the development of *flowcatchR* has not been formally published yet. A manuscript has been submitted for peer-review. For the time being, users of *flowcatchR* are encouraged to cite it using the output of the `citation` function in the *utils*, as it follows:

```
citation("flowcatchR")
```

```
##
## To cite package 'flowcatchR' in publications use:
##
##   Federico Marini (2015). flowcatchR: Tools to analyze in vivo
##   microscopy imaging data focused on tracking flowing blood cells. R
##   package version 1.8.0. https://github.com/federicomarini/flowcatchR
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {flowcatchR: Tools to analyze in vivo microscopy imaging data focused on
## tracking flowing blood cells},
##     author = {Federico Marini},
##     year = {2015},
##     note = {R package version 1.8.0},
##     url = {https://github.com/federicomarini/flowcatchR},
##   }
```

3 Processing overview

flowcatchR works primarily with sets of fluorescent time-lapse images, where the particles of interest are marked with a fluorescent label (e.g., red for blood platelets, green for leukocytes). Although different entry spots are provided (such as the coordinates of identified points in each frame via tab delimited files), we will illustrate the characteristics of the package starting from the common protocol starting point. In this case, we have a set of 20 frames derived from an intravital microscopy acquisition, which for the sake of practicality were already registered to reduce the unwanted specimen movements (Fiji [8] was used for this purpose).

```
library("flowcatchR")
```

```
## Loading required package:  EBImage
```

```
data("MesenteriumSubset")
```

```
# printing summary information for the MesenteriumSubset object
MesenteriumSubset
```

```
## Frames
##   colorMode    : Color
##   storage.mode : double
##   dim          : 271 131 3 20
##   frames.total : 60
##   frames.render: 20
##
## imageData(object)[1:5,1:6,1,1]
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]
```

```
## [1,] 0.1647059 0.2117647 0.1882353 0.1803922 0.1607843 0.1333333
## [2,] 0.2352941 0.1882353 0.1803922 0.1568627 0.1411765 0.1372549
## [3,] 0.2352941 0.2000000 0.1764706 0.1490196 0.1333333 0.1333333
## [4,] 0.2352941 0.2117647 0.1764706 0.1529412 0.1411765 0.1411765
## [5,] 0.2313725 0.2078431 0.1725490 0.1411765 0.1294118 0.1411765
##
## Channel(s): all
```

To obtain the set of trajectories identified from the analysis of the loaded frames, a very compact one-line command is all that is needed:

```
# one command to seize them all :)
fullResults <- kinematics(trajectories(particles(channel.Frames(MesenteriumSubset,"red"))))
```

On a MAC OS X machine equipped with 2.8 Ghz Intel Core i7 processor and 16 GB RAM, the execution of this command takes 2.32 seconds to run (tests performed with the *microbenchmark*).

The following sections will provide additional details to the operations mentioned above, with more also on the auxiliary functions that are available in *flowcatchR*.

4 Image acquisition

A set of images is acquired, after a proper microscopy setup has been performed. This includes for example a careful choice of spatial and temporal resolution; often a compromise must be met to have a good frame rate and a good SNR to detect the particles in the single frames. For a good review on the steps to be taken, please refer to Meijering's work [4, 7].

flowcatchR provides an S4 class that can store the information of a complete acquisition, namely *Frames*. The *Frames* class extends the *Image* class, defined in the *EBImage* package, and thus exploits the multi-dimensional array structures of the class. The locations of the images are stored as `dimnames` of the *Frames* object. To construct a *Frames* object from a set of images, the `read.Frames` function is used:

```
# initialization
fullData <- read.Frames(image.files="/path/to/folder/containing/images/", nframes=100)
# printing summary information for the Frames object
fullData
```

`nframes` specifies the number of frames that will constitute the *Frames* object, whereas `image.files` is a vector of character strings with the full location of the (raw) images, or the path to the folder containing them (works automatically if images are in TIFF/JPG/PNG format). In this case we just loaded the full dataset, but for the demonstrational purpose of this vignette, we will proceed with the subset available in the `MesenteriumSubset` object, which we previously loaded in Section 3.

It is possible to inspect the images composing a *Frames* object with the function `inspect.Frames` (Fig.1).

```
inspect.Frames(MesenteriumSubset, nframes=9, display.method="raster")
```

By default, `display.method` is set to "browser", as in the *EBImage* function display. This opens up a window in the predefined browser (e.g. Mozilla Firefox), with navigable frames (arrows on the top left corner). For the vignette, we will set it to `raster`, for viewing them as raster graphics using R's native functions.

Importantly, these image sets were already registered and rotated in such a way that the overall direction of the movement of interest flows from left to right, as a visual aid and also to fit with some assumptions that

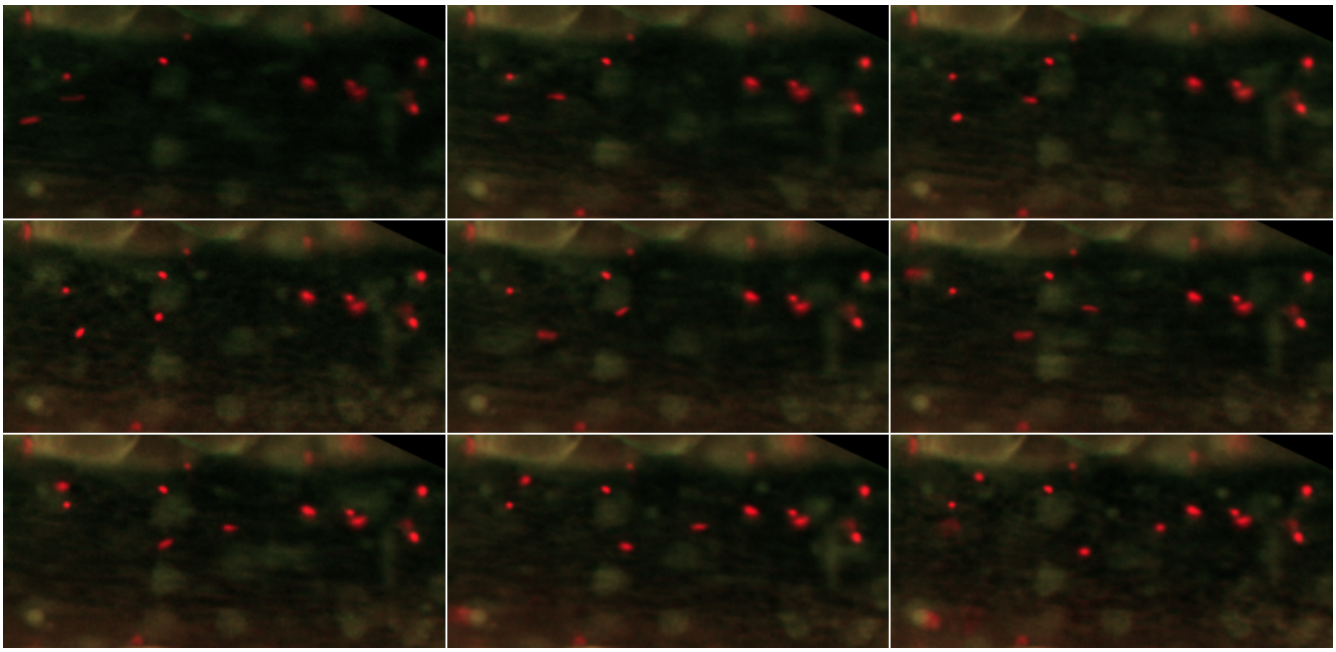

Figure 1: *The first 9 frames of the MesenteriumSubset dataset. The red channel stores information about platelets, while the green channel is dedicated to leukocytes*

will be done in the subsequent step of particle tracking. To register the images, we recommend the general purpose tools offered by suites such as ImageJ/Fiji [9, 8].

For the following steps, we will focus on the information contained in the red channel, corresponding in this case to blood platelets. We do so by calling the `channel.Frames` function:

```
plateletsMesenterium <- channel.Frames(MesenteriumSubset, mode="red")
plateletsMesenterium
```

```
## Frames
##   colorMode    : Grayscale
##   storage.mode : double
##   dim          : 271 131 20
##   frames.total : 20
##   frames.render: 20
##
## imageData(object)[1:5,1:6,1]
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]
## [1,] 0.1647059 0.2117647 0.1882353 0.1803922 0.1607843 0.1333333
## [2,] 0.2352941 0.1882353 0.1803922 0.1568627 0.1411765 0.1372549
## [3,] 0.2352941 0.2000000 0.1764706 0.1490196 0.1333333 0.1333333
## [4,] 0.2352941 0.2117647 0.1764706 0.1529412 0.1411765 0.1411765
## [5,] 0.2313725 0.2078431 0.1725490 0.1411765 0.1294118 0.1411765
##
## Channel(s): red
```

This creates another instance of the class *Frames*, and we inspect it in its first 9 frames (Fig.2).

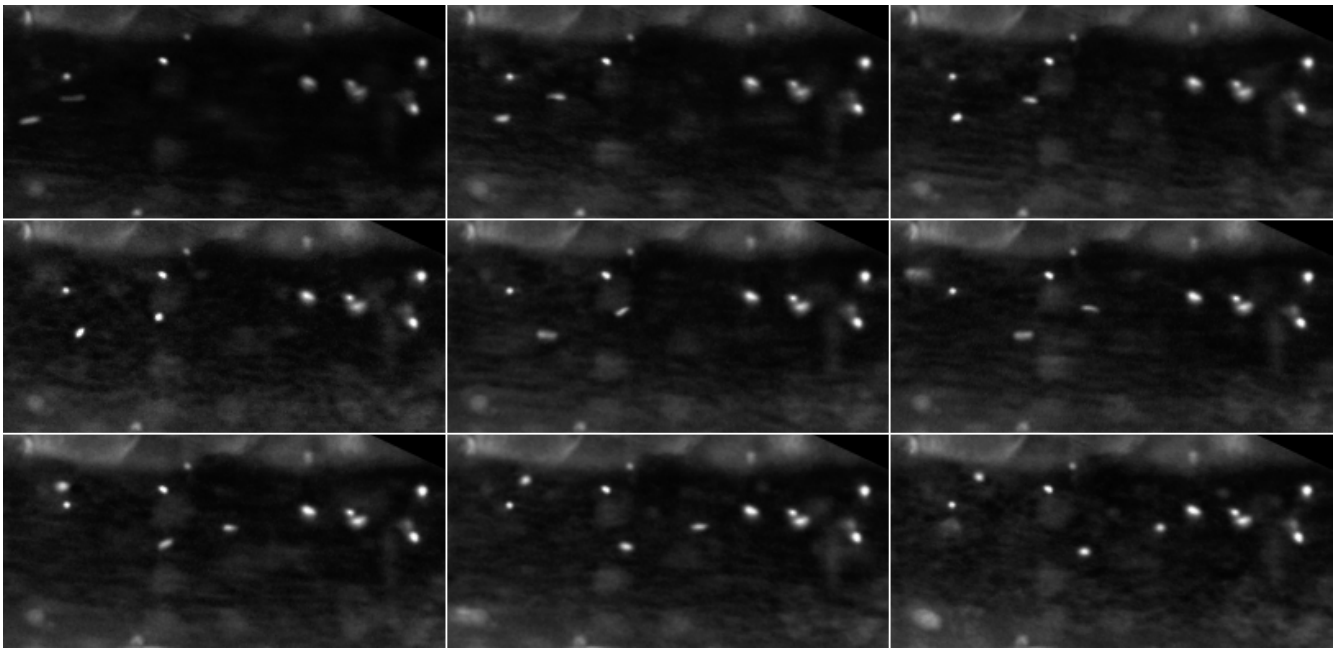

Figure 2: *The first 9 frames of the MesenteriumSubset dataset, just displaying the GrayScale signal for the red channel stored in plateletsMesenterium (for the thrombocytes)*

```
inspect.Frames(plateletsMesenterium, nframes=9, display.method="raster")
```

5 Image preprocessing and analysis

Steps such as denoising, smoothing and morphological operations (erosion/dilation, opening/closing) can be performed thanks to the general functions provided by *EBImage*. *flowcatchR* offers a wrapper around a series of operations to be applied to all images in a *Frames* object. The function `preprocess.Frames` is called via the following command:

```
preprocessedPlatelets <- preprocess.Frames(plateletsMesenterium,
                                    brush.size=3, brush.shape="disc",
                                    at.offset=0.15, at.wwidth=10, at.wheight=10,
                                    kern.size=3, kern.shape="disc",
                                    ws.tolerance=1, ws.radius=1)
preprocessedPlatelets
```

```
## Frames
##   colorMode    : Grayscale
##   storage.mode : double
##   dim          : 271 131 20
##   frames.total : 20
##   frames.render: 20
##
## imageData(object)[1:5,1:6,1]
##      [,1] [,2] [,3] [,4] [,5] [,6]
```

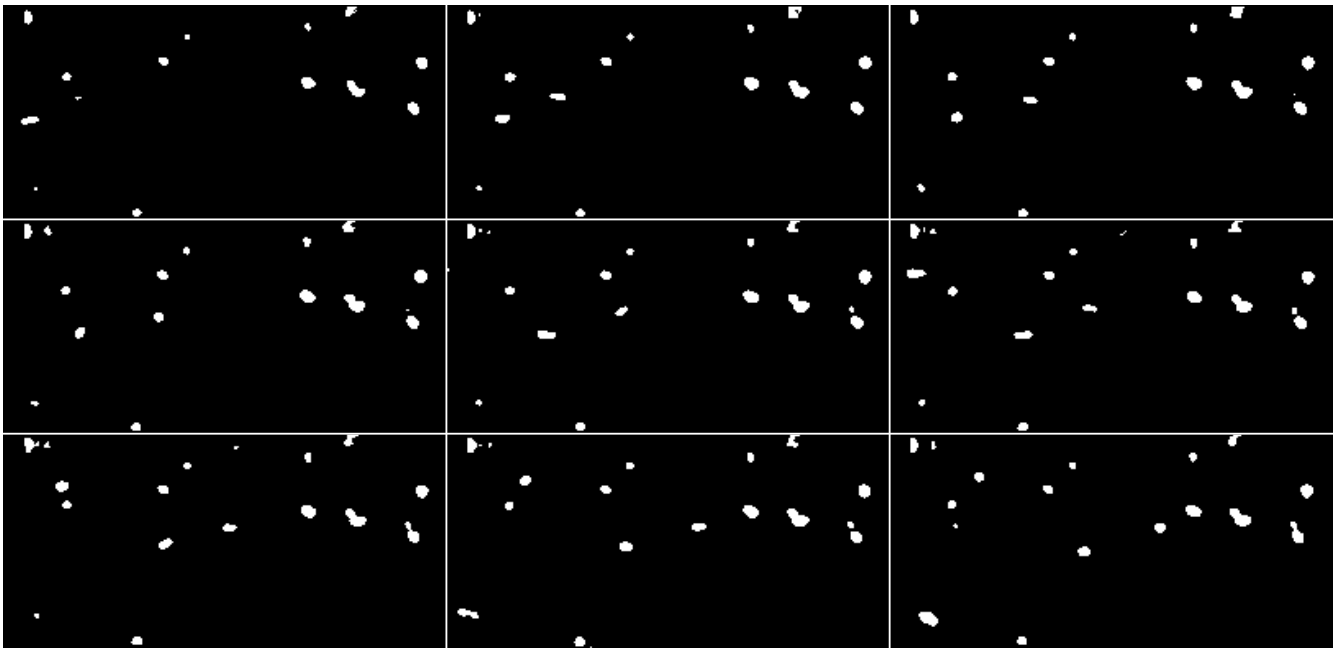

Figure 3: *The first 9 frames after preprocessing of the MesenteriumSubset dataset. The binarized image shows the detected objects after thresholding.*

```
## [1,]    0    0    0    0    0    0
## [2,]    0    0    0    0    0    0
## [3,]    0    0    0    0    0    0
## [4,]    0    0    0    0    0    0
## [5,]    0    0    0    0    0    0
##
## Channel(s): red
```

The result of this is displayed in Fig.3. For a detailed explanation of the parameters to better tweak the performances of this segmentation step, please refer to the help of `preprocess.Frames`. To obtain an immediate feedback about the effects of the operations performed in the full preprocessing phase, we can call again `inspect.Frames` on the *Frames* of segmented images (Fig.3).

```
inspect.Frames(preprocessedPlatelets, nframes=9, display.method="raster")
```

The frames could be cropped, if e.g. it is needed to remove background noise that might be present close to the edges. This is done with the function `crop.Frames`.

```
croppedFrames <- crop.Frames(plateletsMesenterium,
                     cutLeft=6, cutRight=6,
                     cutUp=3, cutDown=3,
                     testing=FALSE)
croppedFrames
```

```
## Frames
##   colorMode    : Grayscale
##   storage.mode : double
##   dim          : 260 126 20
```

```
##   frames.total : 20
##   frames.render: 20
##
## imageData(object)[1:5,1:6,1]
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]
## [1,] 0.1803922 0.1568627 0.1372549 0.1372549 0.1333333 0.1176471
## [2,] 0.2039216 0.1843137 0.1490196 0.1254902 0.1215686 0.1215686
## [3,] 0.1843137 0.1764706 0.1568627 0.1294118 0.1176471 0.1019608
## [4,] 0.1921569 0.1568627 0.1529412 0.1333333 0.1254902 0.1333333
## [5,] 0.2000000 0.1647059 0.1490196 0.1450980 0.1333333 0.1411765
##
## Channel(s): red
```

If `testing` is set to true, the function just displays the first cropped frame, to get a feeling whether the choice of parameters was adequate. Similarly, for the function `rotate.Frames` the same behaviour is expected, whereas the rotation in degrees is specified by the parameter `angle`.

```
rotatedFrames <- rotate.Frames(plateletsMesenterium, angle=30)
rotatedFrames
```

```
## Frames
##   colorMode    : Grayscale
##   storage.mode : double
##   dim          : 300 249 20
##   frames.total : 20
##   frames.render: 20
##
## imageData(object)[1:5,1:6,1]
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    0    0    0    0    0    0
## [2,]    0    0    0    0    0    0
## [3,]    0    0    0    0    0    0
## [4,]    0    0    0    0    0    0
## [5,]    0    0    0    0    0    0
##
## Channel(s): red
```

If desired, it is possible to select just a subset of the frames belonging to a *Frames*. This can be done via the `select.Frames` function:

```
subsetFrames <- select.Frames(plateletsMesenterium,
                    framesToKeep=c(1:10,14:20))
subsetFrames
```

```
## Frames
##   colorMode    : Grayscale
##   storage.mode : double
##   dim          : 271 131 17
##   frames.total : 17
##   frames.render: 17
##
## imageData(object)[1:5,1:6,1]
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]
## [1,] 0.1647059 0.2117647 0.1882353 0.1803922 0.1607843 0.1333333
## [2,] 0.2352941 0.1882353 0.1803922 0.1568627 0.1411765 0.1372549
```

```
## [3,] 0.2352941 0.2000000 0.1764706 0.1490196 0.1333333 0.1333333
## [4,] 0.2352941 0.2117647 0.1764706 0.1529412 0.1411765 0.1411765
## [5,] 0.2313725 0.2078431 0.1725490 0.1411765 0.1294118 0.1411765
##
## Channel(s): red
```

If required, the user can decide to perform a normalization step (via `normalizeFrames`), to correct for systematic variations in the acquisition conditions, in case the overall intensity levels change, e.g., when the acquisition spans long time scales. In this case, the median of the intensity sums is chosen as a scaling factor.

```
normFrames <- normalizeFrames(plateletsMesenterium,normFun = "median")
```

The user can choose any combination of the operations in order to segment the images provided as input, but `preprocess.Frames` is a very convenient high level function for proceeding in the workflow. It is also possible, as it was shown in the introductory one-liner, to call just `particles` on the raw *Frames* object. In this latter case, `particles` computes the preprocessed *Frames* object according to default parameters. Still, in either situation, the output for this step is an object of the *ParticleSet* class.

```
platelets <- particles(plateletsMesenterium, preprocessedPlatelets)
platelets
```

```
## Computing features in parallel...
## Done!
## An object of the ParticleSet class.
##
## Set of particles for 20 images
##
## Displaying a subset of the features of the 14 particles found in the first image...
##   cell.0.m.cx cell.0.m.cy cell.0.m.majoraxis cell.0.m.eccentricity
## 1   186.70833   47.937500           8.916405             0.6353715
## 2   256.19048   35.857143           7.746554             0.4606665
## 3   251.09524   63.523810           8.552466             0.6843186
## 4    15.79412    8.058824           8.419655             0.7892764
## 5   215.54688   51.828125          13.694783             0.8788344
##   cell.0.m.theta cell.0.s.area cell.0.s.perimeter cell.0.s.radius.mean
## 1      0.3380463            48                 20             3.487042
## 2      0.9165252            42                 19             3.194559
## 3      0.9370618            42                 19             3.247531
## 4      1.4806734            34                 18             2.847459
## 5      0.7172299            64                 28             4.308530
##
## Particles identified on the red channel
```

The `particles` leverages on the multi-core architecture of the systems where the analysis is run, and this is implemented via *BiocParallel* (updated since Version 1.0.3).

As it can be seen from the summary information, each *ParticleSet* stores the essential information on all particles that were detected in the original images, alongside with a complete set of features, which are computed by integrating the information from both the raw and the segmented frames.

A *ParticleSet* can be seen as a named list, where each element is a `data.frame` for a single frame, and the image source is stored as `names` to help backtracking the operations performed, and the slot `channel`is retained as selected in the initial steps.

It is possible to filter out particles according to their properties, such as area, shape and eccentricity. This is

possible with the function `select.particles`. The current implementation regards only the surface extension, but any additional feature can be chosen and adopted to restrict the number of candidate particles according to particular properties which are expected and/or to remove potential noise that went through the preprocessing phase.

```
selectedPlatelets <- select.particles(platelets, min.area=3)
selectedPlatelets
```

```
## Filtering the particles...
## An object of the ParticleSet class.
##
## Set of particles for 20 images
##
## Displaying a subset of the features of the 14 particles found in the first image...
##   cell.0.m.cx cell.0.m.cy cell.0.m.majoraxis cell.0.m.eccentricity
## 1   186.70833   47.937500           8.916405             0.6353715
## 2   256.19048   35.857143           7.746554             0.4606665
## 3   251.09524   63.523810           8.552466             0.6843186
## 4    15.79412    8.058824           8.419655             0.7892764
## 5   215.54688   51.828125          13.694783             0.8788344
##   cell.0.m.theta cell.0.s.area cell.0.s.perimeter cell.0.s.radius.mean
## 1      0.3380463            48                 20             3.487042
## 2      0.9165252            42                 19             3.194559
## 3      0.9370618            42                 19             3.247531
## 4      1.4806734            34                 18             2.847459
## 5      0.7172299            64                 28             4.308530
##
## Particles identified on the red channel
```

This step can be done iteratively, with the help of the function `add.contours`. If called with the parameter `mode` set to `particles`, then it will automatically generate a *Frames* object, with the contours of all particles drawn around the objects that passed the segmentation (and filtering) step (Fig.4).

```
paintedPlatelets <- add.contours(raw.frames=MesenteriumSubset,
                                 binary.frames=preprocessedPlatelets,
                                 mode="particles")
inspect.Frames(paintedPlatelets, nframes=9, display.method="raster")
```

To connect the particles from one frame to the other, we perform first the detection of particles on all images. Only in a successive phase, we establish the links between the so identified objects. This topic will be covered in detail in the following section.

6 Particle tracking

To establish the connections between particles, the function to be called is `link.particles`. The algorithm used to perform the tracking itself is an improved version of the original implementation of Sbalzarini and Koumotsakos [5]. To summarize the method, it is a fast and efficient self-initializing feature point tracking algorithm (using the centroids of the objects as reference) [10]. The initial version is based on a particle matching algorithm, approached via a graph theory technique. It allows for appearances/disappearances of particles from the field of view, also temporarily as it happens in case of occlusions and objects leaving the plane of focus.

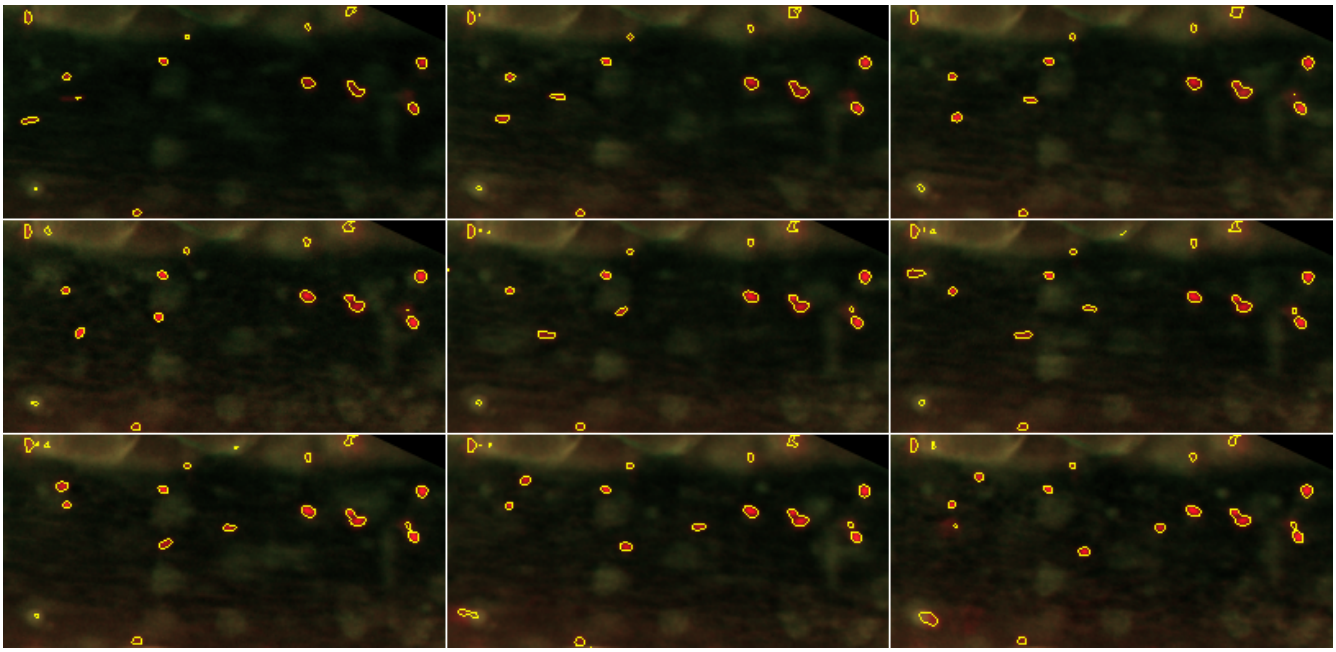

Figure 4: *Particles detected in the first 9 frames are shown in yellow, with their contours defined by the segmentation procedure.*

Our implementation adds to the existing one by redefining the cost function used in the optimization phase of the link assignment. It namely adds two terms, such as intensity variation and area variation, and mostly important implements a function to penalize the movements that are either perpendicular or backwards with respect to the oriented flow of cells. Small unwanted movements, which may be present even after the registration phase, are handled with two jitter terms in a defined penalty function. Multiplicative factors can further influence the penalties given to each term.

In its default value, the penalty function is created via the `penaltyFunctionGenerator`. The user can exploit the parameter values in it to create a custom version of it, to match the particular needs stemming from the nature of the available data and phenomenon under inspection.

```
defaultPenalty <- penaltyFunctionGenerator()
print(defaultPenalty)
```

```
## function (angle, distance)
## {
##     lambda1 * (distance/(1 - lambda2 * (abs(angle)/(pi + epsilon1))))
## }
## <environment: 0x00000000099c8618>
```

As mentioned above, to perform the linking of the particles, we use `link.particles`. Fundamental parameters are `L` and `R`, named as in the original implementation. `L` is the maximum displacement in pixels that a particle is expected to have in two consecutive frames, and `R` is the value for the link range, i.e. the number of future frames to be considered for the linking (typically assumes values between 1 - when no occlusions are known to happen - and 3). An extended explanation of the parameters is in the documentation of the package.

```
linkedPlatelets <- link.particles(platelets,
                                  L=26, R=3,
                                  epsilon1=0, epsilon2=0,
                                  lambda1=1, lambda2=0,
                                  penaltyFunction=penaltyFunctionGenerator(),
                                  include.area=FALSE)
linkedPlatelets
```

```
## An object of the LinkedParticleSet class.
##
## Set of particles for 20 images
##
## Particles are tracked throughout the subsequent 3 frame(s)
##
## Displaying a subset of the features of the 14 particles found in the first image...
##   cell.0.m.cx cell.0.m.cy cell.0.m.majoraxis cell.0.m.eccentricity
## 1   186.70833   47.937500           8.916405             0.6353715
## 2   256.19048   35.857143           7.746554             0.4606665
## 3   251.09524   63.523810           8.552466             0.6843186
## 4    15.79412    8.058824           8.419655             0.7892764
## 5   215.54688   51.828125          13.694783             0.8788344
##   cell.0.m.theta cell.0.s.area cell.0.s.perimeter cell.0.s.radius.mean
## 1      0.3380463            48                 20             3.487042
## 2      0.9165252            42                 19             3.194559
## 3      0.9370618            42                 19             3.247531
## 4      1.4806734            34                 18             2.847459
## 5      0.7172299            64                 28             4.308530
##
## Particles identified on the red channel
```

As it can be seen, `linkedPlatelets` is an object of the *LinkedParticleSet* class, which is a subclass of the *ParticleSet* class.

After inspecting the trajectories (see Section 7) it might be possible to filter a *LinkedParticleSet* class object and subsequently reperform the linking on its updated version (e.g. some detected particles were found to be noise, and thus removed with `select.particles`).

flowcatchR provides functions to export and import the identified particles, in order to offer an additional entry point for tracking and analyzing the trajectories (if particles were detected with other routines) and also to store separately the information per each frame about the objects that were primarily identified.

An example is provided in the lines below, with the functions `export.particles` and `read.particles` :

```
# export to csv format
export.particles(platelets, dir="/path/to/export/folder/exportParticleSet/")
# re-import the previously exported, in this case
importedPlatelets <- read.particles(particle.files="/path/to/export/folder/exportParticleSet/")
```

7 Trajectory analysis

It is possible to extract the trajectories with the correspondent `trajectories` function:

```
trajPlatelets <- trajectories(linkedPlatelets)
trajPlatelets
```

```
## Generating trajectories...
## An object of the TrajectorySet class.
##
## TrajectorySet composed of 25 trajectories
##
## Trajectories cover a range of 20 frames
## Displaying a segment of the first trajectory...
##        xCoord   yCoord trajLabel frame frameobjectID
## 1_1  186.7083 47.93750         1     1             1
## 1_2  186.9649 48.26316         1     2             4
## 1_3  186.8136 48.18644         1     3             2
## 1_4  186.2807 47.70175         1     4             1
## 1_5  186.6897 47.87931         1     5             2
## 1_6  186.8269 48.11538         1     6             2
## 1_7  186.9643 48.30357         1     7             1
## 1_8  186.6207 48.36207         1     8             3
## 1_9  186.3273 48.05455         1     9             3
## 1_10 186.9821 48.19643         1    10             3
##
## Trajectories are related to particles identified on the red channel
```

A *TrajectorySet* object is returned in this case. It consists of a two level list for each trajectory, reporting the `trajectory` as a `data.frame`, the number of points `npoints` (often coinciding with the number of `nframes`, when no gaps `ngaps` are present) and its `ID`. A `keep` flag is used for subsequent user evaluation purposes.

Before proceeding with the actual analysis of the trajectories, it is recommended to evaluate them by visual inspection. *flowcatchR* provides two complementary methods to do this, either plotting them (`plot` or `plot2D.TrajectorySet`) or drawing the contours of the points on the original image (`add.contours`).

By plotting all trajectories in a 2D+time representation, it's possible to have an overview of all trajectories.

The following command gives an interactive 3D (2D+time) view of all trajectories (output is not included in this vignette):

```
plot(trajPlatelets, MesenteriumSubset)
```

The `plot2D.TrajectorySet` focuses on additional information and a different "point of view", but can just display a two dimensional projection of the identified trajectories (Fig.5).

```
plot2D.TrajectorySet(trajPlatelets, MesenteriumSubset)
```

To have more insights on single trajectories, or on a subset of them, `add.contours` offers an additional mode called `trajectories`. Particles are drawn on the raw images with colours corresponding to the trajectory IDs. `add.contours` plots by default all trajectories, but the user can supply a vector of the IDs of interest to override this behaviour.

```
paintedTrajectories <- add.contours(raw.frames=MesenteriumSubset,
                                    binary.frames=preprocessedPlatelets,
                                    trajectoryset=trajPlatelets,
                                    mode="trajectories")
paintedTrajectories
```

```
## Frames
```

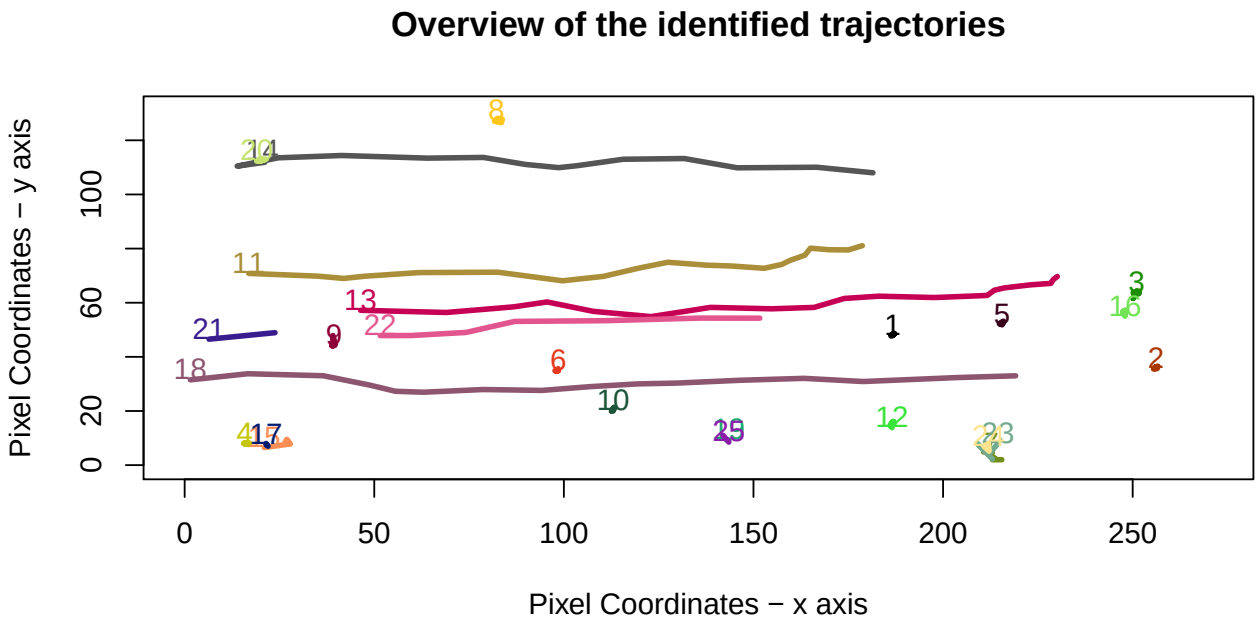

Figure 5: *A 2D "flat" representation of the trajectories, more suitable to give an indication of the global movement*

```
##   colorMode    : Color
##   storage.mode : double
##   dim          : 271 131 3 20
##   frames.total : 60
##   frames.render: 20
##
## imageData(object)[1:5,1:6,1,1]
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]
## [1,] 0.1647059 0.2117647 0.1882353 0.1803922 0.1607843 0.1333333
## [2,] 0.2352941 0.1882353 0.1803922 0.1568627 0.1411765 0.1372549
## [3,] 0.2352941 0.2000000 0.1764706 0.1490196 0.1333333 0.1333333
## [4,] 0.2352941 0.2117647 0.1764706 0.1529412 0.1411765 0.1411765
## [5,] 0.2313725 0.2078431 0.1725490 0.1411765 0.1294118 0.1411765
##
## Channel(s): all
```

As with any other *Frames* object, it is recommended to take a peek at it via the `inspect.Frames` function (Fig.6):

```
inspect.Frames(paintedTrajectories,nframes=9,display.method="raster")
```

To allow for a thorough evaluation of the single trajectories, `export.Frames` is a valid helper, as it creates single images corresponding to each frame in the *Frames* object. We first extract for example trajectory 11 (Fig.7) with the following command:

```
traj11 <- add.contours(raw.frames=MesenteriumSubset,
                       binary.frames=preprocessedPlatelets,
                       trajectoryset=trajPlatelets,
                       mode="trajectories",
```

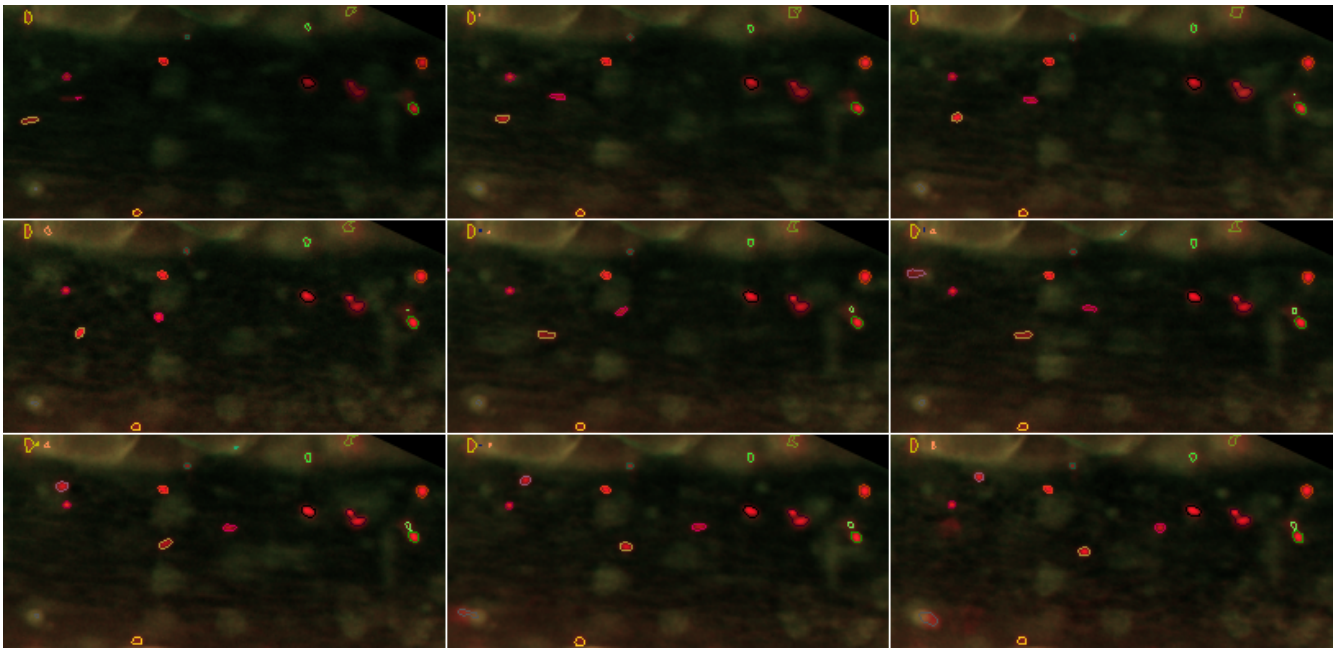

Figure 6: *Particles detected in the first 9 frames are shown this time in colours corresponding to the identified trajectories, again with their contours defined by the segmentation procedure.*

```
                      trajIDs=11)
traj11
inspect.Frames(traj11, nframes=9, display.method="raster")
```

```
## Frames
##   colorMode    : Color
##   storage.mode : double
##   dim          : 271 131 3 20
##   frames.total : 60
##   frames.render: 20
##
## imageData(object)[1:5,1:6,1,1]
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]
## [1,] 0.1647059 0.2117647 0.1882353 0.1803922 0.1607843 0.1333333
## [2,] 0.2352941 0.1882353 0.1803922 0.1568627 0.1411765 0.1372549
## [3,] 0.2352941 0.2000000 0.1764706 0.1490196 0.1333333 0.1333333
## [4,] 0.2352941 0.2117647 0.1764706 0.1529412 0.1411765 0.1411765
## [5,] 0.2313725 0.2078431 0.1725490 0.1411765 0.1294118 0.1411765
##
## Channel(s): all
```

The data for trajectory 11 in the *TrajectorySet* object can be printed to the terminal:

```
trajPlatelets[[11]]
```

```
## $trajectory
##          xCoord   yCoord trajLabel frame frameobjectID
## 11_1   16.89744 70.82051        11     1            11
```

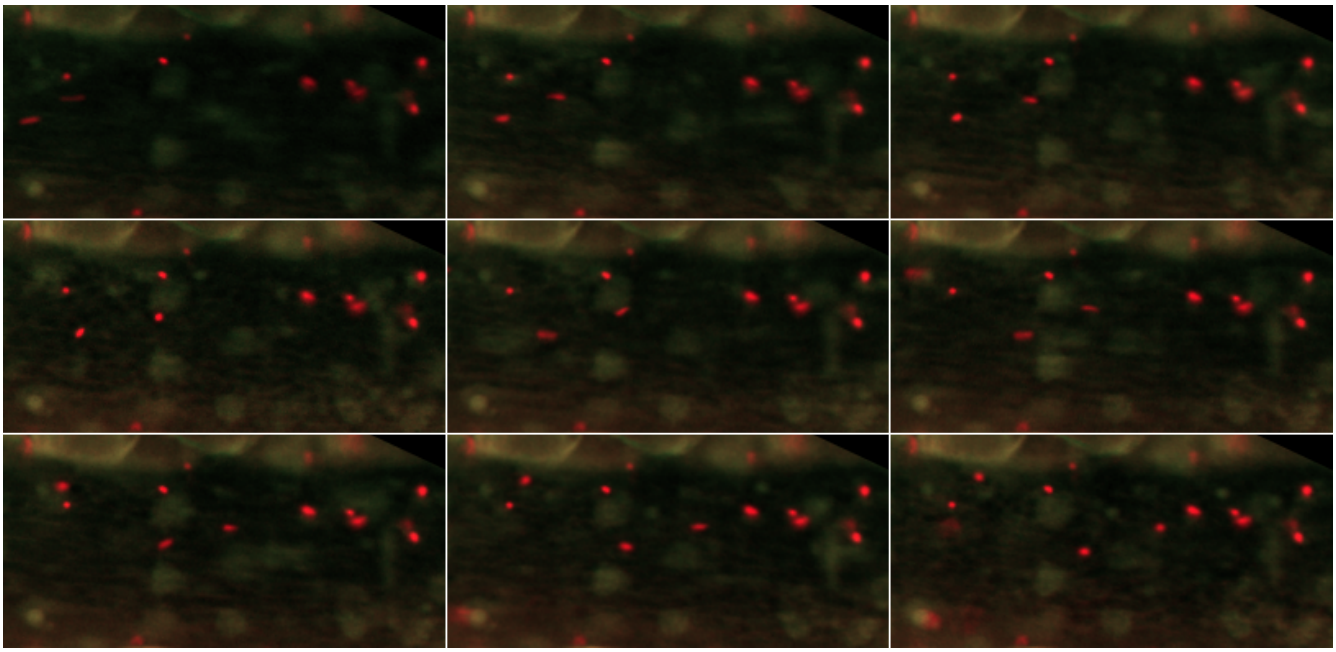

Figure 7: *First 9 frames for trajectory with ID 11, as supplied to the trajIds argument of add.contours*

```
## 11_2   35.00000 69.82051        11     2              5
## 11_3   41.94286 68.97143        11     3              4
## 11_4   47.48485 69.78788        11     4              5
## 11_5   61.68750 71.12500        11     5              6
## 11_6   82.56818 71.27273        11     6              9
## 11_7   99.64103 68.10256        11     7              8
## 11_8  110.37500 69.72500        11     8              5
## 11_9  119.63158 72.73684        11     9              6
## 11_10 127.40541 74.94595        11    10              6
## 11_11 137.57143 73.88095        11    11              5
## 11_12 144.25641 73.56410        11    12              7
## 11_13 152.83333 72.71429        11    13              7
## 11_14 157.47500 74.15000        11    14              4
## 11_15 159.71795 75.69231        11    15              7
## 11_16 163.56410 77.56410        11    16              6
## 11_17 165.02564 80.15385        11    17              2
## 11_18 169.94595 79.56757        11    18              6
## 11_19 175.00000 79.50000        11    19              5
## 11_20 178.71795 81.07692        11    20              7
##
## $npoints
## [1] 20
##
## $nframes
## [1] 20
##
## $ngaps
## [1] 0
```

```
##
## $keep
## [1] NA
##
## $ID
## [1] 11
```

After that, it can also be exported with the following command (the `dir` parameter must be changed accordingly):

```
export.Frames(traj11, dir=tempdir(), nameStub="vignetteTest_traj11",
              createGif=TRUE, removeAfterCreatingGif=FALSE)
```

`export.Frames` offers multiple ways to export - animated gif (if *ImageMagick* is available and installed on the system) or multiple jpeg/png images.

Of course the user might want to singularly evaluate each trajectory that was identified, and this can be done by looping over the trajectory IDs.

```
evaluatedTrajectories <- trajPlatelets

for(i in 1:length(trajPlatelets))
{
  paintedTraj <- add.contours(raw.frames=MesenteriumSubset,
                              binary.frames=preprocessedPlatelets,
                              trajectoryset=trajPlatelets,
                              mode="trajectories",
                              col="yellow",
                              trajIDs=i)
  export.Frames(paintedTraj,
                nameStub=paste0("vignetteTest_evaluation_traj_oneByOne_",i),
                createGif=TRUE, removeAfterCreatingGif=TRUE)
  ### uncomment the code below to perform the interactive evaluation of the single trajectories

  #   cat("Should I keep this trajectory? --- 0: NO, 1:YES --- no other values allowed")
  #   userInput <- readLines(n=1L)
  #   ## if neither 0 nor 1, do not update
  #   ## otherwise, this becomes the value for the field keep in the new TrajectoryList
  #   evaluatedTrajectories@.Data[[i]]£keep <- as.logical(as.numeric(userInput))
}
```

Always using trajectory 11 as example, we would set `evaluatedTrajectories[[11]]$keep` to `TRUE`, since the trajectory was correctly identified, as we just checked.

Once all trajectories have been selected, we can proceed to calculate (a set of) kinematic parameters, for a single or all trajectories in a *TrajectorySet* object. The function `kinematics` returns the desired output, respectively a *KinematicsFeatures* object, a *KinematicsFeaturesSet*, a single value or a vector (or list, if not coercible to vector) of these single values (one parameter for each trajectory).

```
allKinematicFeats.allPlatelets <- kinematics(trajPlatelets,
                                             trajectoryIDs=NULL, # will select all trajectory IDs
                                             acquisitionFrequency=30, # value in milliseconds
                                             scala=50, # 1 pixel is equivalent to ... micrometer
                                             feature=NULL) # all kinematic features available
```

```
## Warning in extractKinematics.traj(trajectoryset, i, acquisitionFrequency = acquisitionFrequency,
:  The trajectory with ID 17 had 3 or less points, no features were computed.

## Warning in extractKinematics.traj(trajectoryset, i, acquisitionFrequency = acquisitionFrequency,
:  The trajectory with ID 19 had 3 or less points, no features were computed.

## Warning in extractKinematics.traj(trajectoryset, i, acquisitionFrequency = acquisitionFrequency,
:  The trajectory with ID 21 had 3 or less points, no features were computed.

## Warning in extractKinematics.traj(trajectoryset, i, acquisitionFrequency = acquisitionFrequency,
:  The trajectory with ID 25 had 3 or less points, no features were computed.
```

As it is reported from the output, the function raises a warning for trajectories which have 3 or less points, as they might be spurious detections. In such cases, no kinematic features are computed.

```
allKinematicFeats.allPlatelets

## An object of the KinematicsFeaturesSet class.
##
## KinematicsFeaturesSet composed of 25  KinematicsFeatures objects
##
## Available features (shown for the first trajectory):
##  [1] "delta.x"                    "delta.t"
##  [3] "delta.v"                    "totalTime"
##  [5] "totalDistance"              "distStartToEnd"
##  [7] "curvilinearVelocity"        "straightLineVelocity"
##  [9] "linearityForwardProgression" "trajMSD"
## [11] "velocityAutoCorr"           "instAngle"
## [13] "directChange"               "dirAutoCorr"
## [15] "paramsNotComputed"
##
## Curvilinear Velocity: 0.009970094
## Total Distance: 5.682953
## Total Time: 570
##
## Average values (calculated on 4 trajectories where parameters were computed)
## Average Curvilinear Velocity: 0.1146995
## Average Total Distance: 48.49261
## Average Total Time: 484.2857
```

A summary for the returned object (in this case a *KinematicsFeaturesSet*) shows some of the computed parameters. By default, information about the first trajectory is reported in brief, and the same parameters are evaluated on average across the selected trajectories. The true values can be accessed in this case for each trajectory by the subset operator for lists (`[[]]`), followed by the name of the kinematic feature (e.g., `$totalDistance`).

A list of all available parameters is printed with an error message if the user specifies an incorrect name, such as here:

```
allKinematicFeats.allPlatelets <- kinematics(trajPlatelets, feature="?")

## Available features to compute are listed here below.
##           Please select one among delta.x, delta.t, delta.v, totalTime,
##           totalDistance, distStartToEnd, curvilinearVelocity,
##           straightLineVelocity, linearityForwardProgression, trajMSD,
##           velocityAutoCorr, instAngle, directChange or dirAutoCorr
```

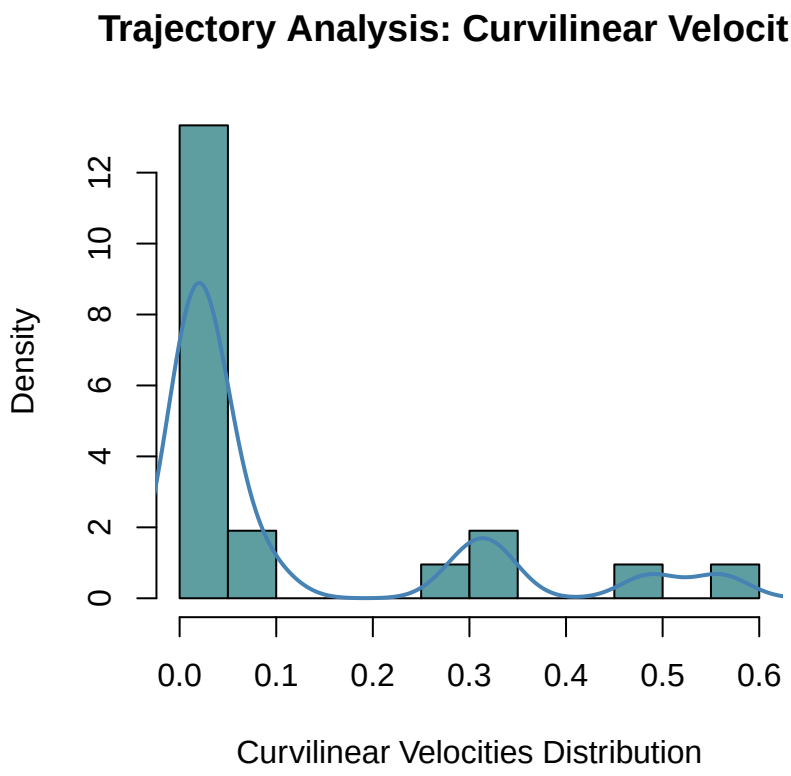

Figure 8: *Histogram of the curvilinear velocities for all trajectories identified in the MesenteriumSubset dataset*

When asking for a single parameter, the value returned is structured in a vector, such that it is straightforward to proceed with further analysis, e.g. by plotting the distribution of the curvilinear velocities (Fig.8).

```
allVelocities <- kinematics(trajPlatelets, feature="curvilinearVelocity")

hist(allVelocities, breaks=10, probability=TRUE, col="cadetblue",
     xlab="Curvilinear Velocities Distribution",
     main="Trajectory Analysis: Curvilinear Velocities")
lines(density(allVelocities, na.rm=TRUE), col="steelblue", lwd=2)
```

For this code chunk, we are suppressing the warning messages, as they would be exactly the same as in the former where all features were computed for each trajectory.

8 Interactive tools for a user-friendly workflow solution

To enhance the *Frames* objects and deliver an immediate feedback to the user, the function `snap` leverages on both the raw and binary *Frames*, as well as on the corresponding *ParticleSet* and *TrajectorySet* objects. It integrates the information available in all the mentioned objects, and it plots a modified instance of the *Frames* object, identifying the particles closest to the mouse click, and showing additional trajectory-related information, such as the trajectory ID and the instantaneous velocity of the cell. The function can be called as in the command below:

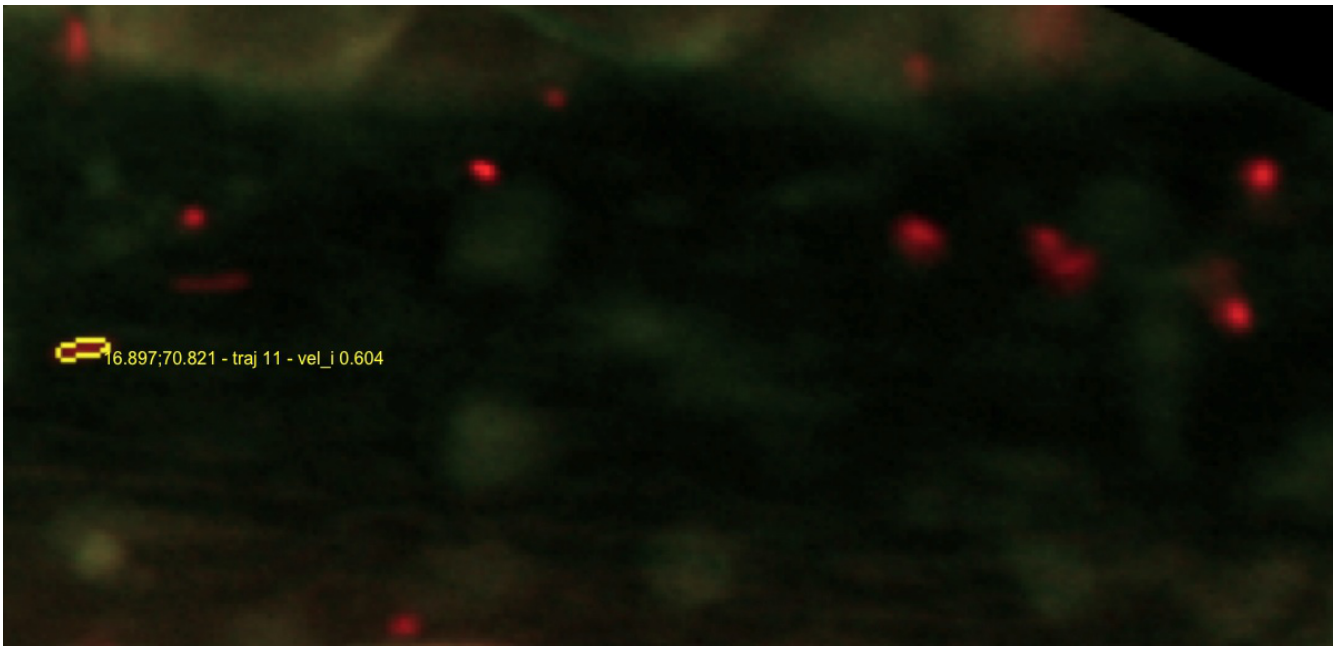

Figure 9: *Output generated by the snap function, where the user wanted to identify the particle and additionally display the trajectory information (ID, instantaneous velocity) on it.*

```
snap(MesenteriumSubset,preprocessedPlatelets,
     platelets,trajPlatelets,
     frameID = 1,showVelocity = T)
```

An example output for the `snap` is shown below in Fig.9, where the information (trajectory ID, as well as the velocity in the selected frame) is shown in yellow to offer a good contrast with the fluorescent image.

8.1 The `shinyFlow` Shiny Application

Additionally, since Version 1.0.3, *flowcatchR* delivers `shinyFlow`, a Shiny Web Application ([11]), which is built on the backbone of the analysis presented in this vignette, and is portable across all main operating systems. The user is thus invited to explore datasets and parameters with immediate reactive feedback, that can enable better understanding of the effects of single steps and changes in the workflow.

To launch the Shiny App, use the command below to open an external window either in the browser or in the IDE (such as RStudio):

```
shinyFlow()
```

8.2 flowcatchR in Jupyter notebooks

A further integration are a number of Jupyter/IPython notebooks ([12]), as a way to provide easy reproducibility as well as communication of results, by combining plain text, commands and output in single documents. The R kernel used on the back-end was developed by Thomas Kluyver (https://github.com/takluyver/IRkernel), and instructions for the installation are available at the Github repository website. The notebooks are available in the installation folder of the package *flowcatchR*, which can be found with the command below.

```
list.files(system.file("extdata",package = "flowcatchR"),pattern = "*.ipynb")
```

```
## [1] "template_DetectionOfTransmigrationEvents.ipynb"
## [2] "template_flowcatchR_vignetteSummary.ipynb"
```

The notebooks are provided as template for further steps in the analysis. The user is invited to set up the IPython notebook framework as explained on the official website for the project (http://ipython.org/notebook.html). As of February, 3rd 2015, the current way to obtain the Jupyter environment is via the `3.0.dev` version, available via Github (https://github.com/ipython/ipython). The notebooks can be opened and edited by navigating to their location while the IPython notebook server is running; use the following command in the shell to launch it:

```
$ ipython notebook
```

Alternatively, these documents can be viewed with the `nbviewer` tool, available at http://nbviewer.ipython.org/.

9 flowcatchR in Docker containers

flowcatchR is now (as of September 2015) available also in Docker images that are the components of the `dockerflow` proposal (https://github.com/federicomarini/dockerflow). This includes:

- `flowstudio` - https://github.com/federicomarini/flowstudio, a command-line/IDE interface to RStudio where *flowcatchR* and its dependencies are preinstalled
- `flowshiny` - https://github.com/federicomarini/flowshiny a Shiny Server running the `shinyFlow` web application
- `flowjupy` - https://github.com/federicomarini/flowjupy, a Jupyter Notebook interface

These three images can be run simultaneously, provided the system where the containers are running supports the `docker-compose` tool. For more information on how to install the single components, please refer to their repositories.

10 Supplementary information

For more information on the method adapted for tracking cells, see Sbalzarini and Koumotsakos (2005) [5]. For additional details regarding the functions of *flowcatchR*, please consult the documentation or write an email to `marinif@uni-mainz.de`.

Due to space limitations, the complete dataset for the acquired frames used in this vignette is not included as part of the *flowcatchR* package. If you would like to get access to it, you can write an email to `marinif@uni-mainz.de`.

11 Acknowledgements

This package was developed at the Institute of Medical Biostatistics, Epidemiology and Informatics at the University Medical Center, Mainz (Germany), with the financial support provided by the TRP-A15 Translational Research Project grant.

flowcatchR incorporates suggestions and feedback from the wet-lab biology units operating at the Center for Thrombosis and Hemostasis (CTH), in particular Sven Jäckel and Kerstin Jurk. Sven Jäckel also provided us with the sample acquisition which is available in this vignette.

We would like to thank the members of the Biostatistics division for valuable discussions, and additionally Isabella Zwiener for contributing to the first ideas on the project.

12 Session Information

This vignette was generated using the following package versions:

```
toLatex(sessionInfo())
```

- R version 3.3.1 (2016-06-21), `x86_64-w64-mingw32`
- Locale: `LC_COLLATE=C`, `LC_CTYPE=English_United States.1252`, `LC_MONETARY=English_United States.1252`, `LC_NUMERIC=C`, `LC_TIME=English_United States.1252`
- Base packages: base, datasets, grDevices, graphics, methods, stats, utils
- Other packages: EBImage 4.16.0, flowcatchR 1.8.0, knitr 1.14
- Loaded via a namespace (and not attached): BiocGenerics 0.20.0, BiocParallel 1.8.0, BiocStyle 2.2.0, R6 2.2.0, Rcpp 0.12.7, abind 1.4-5, colorRamps 2.3, digest 0.6.10, evaluate 0.10, fftwtools 0.9-7, formatR 1.4, grid 3.3.1, highr 0.6, htmltools 0.3.5, htmlwidgets 0.7, httpuv 1.3.3, jpeg 0.1-8, jsonlite 1.1, lattice 0.20-34, locfit 1.5-9.1, magrittr 1.5, mime 0.5, parallel 3.3.1, png 0.1-7, rgl 0.96.0, shiny 0.14.1, snow 0.4-2, stringi 1.1.2, stringr 1.1.0, tiff 0.1-5, tools 3.3.1, xtable 1.8-2

References

[1] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2014. URL: http://www.R-project.org/.

[2] Robert C Gentleman, Vincent J. Carey, Douglas M. Bates, and others. Bioconductor: Open software development for computational biology and bioinformatics. *Genome Biology*, 5:R80, 2004. URL: http://genomebiology.com/2004/5/10/R80.

[3] Grégoire Pau, Florian Fuchs, Oleg Sklyar, Michael Boutros, and Wolfgang Huber. EBImage–an R package for image processing with applications to cellular phenotypes. *Bioinformatics*, 26(7):979–81, April 2010. URL: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=2844988&tool=pmcentrez&rendertype=abstract, doi:10.1093/bioinformatics/btq046.

[4] Erik Meijering and Ihor Smal. Time-lapse imaging. In *Microscope Image Processing*, pages 401–440. Academic Press, 2008. URL: http://www.imagescience.org/meijering/publications/download/tlmi2008.pdf.

[5] I F Sbalzarini and P Koumoutsakos. Feature point tracking and trajectory analysis for video imaging in cell biology. *Journal of structural biology*, 151(2):182–95, August 2005. URL: http://www.ncbi.nlm.nih.gov/pubmed/16043363, doi:10.1016/j.jsb.2005.06.002.

[6] JB Beltman, AFM Marée, and RJ de Boer. Analysing immune cell migration. *Nature Reviews Immunology*, 9(11):789–798, 2009. URL: http://dx.doi.org/10.1038/nri2638http://www.nature.com/nri/journal/v9/n11/abs/nri2638.html, doi:10.1038/nri2638.

[7] Erik Meijering, Oleh Dzyubachyk, and Ihor Smal. Methods for cell and particle tracking. *Methods in enzymology*, 504(February):183–200, January 2012. URL: http://www.ncbi.nlm.nih.gov/pubmed/22264535, doi:10.1016/B978-0-12-391857-4.00009-4.

[8] Johannes Schindelin, I Arganda-Carreras, and Erwin Frise. Fiji: an open-source platform for biological-image analysis. *Nature methods*, 9(7):676–682, 2012. URL: http://www.nature.com/nmeth/journal/v9/n7/full/nmeth.2019.html%3FWT.ec_id%3DNMETH-201207, doi:10.1038/nmeth.2019.

[9] Caroline a Schneider, Wayne S Rasband, and Kevin W Eliceiri. NIH Image to ImageJ: 25 years of image analysis. *Nature Methods*, 9(7):671–675, June 2012. URL: http://www.nature.com/doifinder/10.1038/nmeth.2089, doi:10.1038/nmeth.2089.

[10] Nicolas Chenouard, Ihor Smal, Fabrice de Chaumont, Martin Maška, Ivo F Sbalzarini, Yuanhao Gong, Janick Cardinale, Craig Carthel, Stefano Coraluppi, Mark Winter, Andrew R Cohen, William J Godinez, Karl Rohr, Yannis Kalaidzidis, Liang Liang, James Duncan, Hongying Shen, Yingke Xu, Klas E G Magnusson, Joakim Jaldén, Helen M Blau, Perrine Paul-Gilloteaux, Philippe Roudot, Charles Kervrann, François Waharte, Jean-Yves Tinevez, Spencer L Shorte, Joost Willemse, Katherine Celler, Gilles P van Wezel, Han-Wei Dan, Yuh-Show Tsai, Carlos Ortiz de Solórzano, Jean-Christophe Olivo-Marin, and Erik Meijering. Objective comparison of particle tracking methods. *Nature methods*, 11(3):281–9, March 2014. URL: http://www.ncbi.nlm.nih.gov/pubmed/24441936, doi:10.1038/nmeth.2808.

[11] RStudio, Inc. *Easy web applications in R.*, 2013. URL: http://www.rstudio.com/shiny/.

[12] Fernando Pérez and Brian E. Granger. IPython: a system for interactive scientific computing. *Computing in Science and Engineering*, 9(3):21–29, May 2007. URL: http://ipython.org, doi:10.1109/MCSE.2007.53.