FM-Track: Feature-based fiducial marker tracking software for applications in cell mechanics

0. Abstract:

Research methods in mechanobiology that involve tracking the deformation of fiducial markers in the vicinity of a cell are increasing in popularity. Here we present a software called FM-Track, a software to facilitate feature-based particle tracking tailored to applications in cell mechanics. FM-Track contains functions for pre-processing images, running fiducial marker tracking, and simple post-processing and visualization. We expect that FM-Track will help researchers in mechanobiology and related fields by providing straightforward and extensible software written in the Python language.

1. Motivation and Significance:

At present, there is substantial research dedicated to understanding how cells interact mechanically with their local environment. One major class of methods for doing this involves either intentionally placing or identifying natural fiducial markers, and tracking the motion of these fiducial markers in the vicinity of a given cell. For example, three-dimensional Traction Force Microscopy (TFM) is an experimental technique that has gained substantial popularity in recent years. In TFM, cells and μm-scale fluorescent beads are seeded in gels and the observed deformation of the fluorescent beads under different biologically relevant conditions is used to infer how the cell is mechanically interacting with the local environment. However, widespread adoption of TFM, and related methods, is limited in part because the software required for successful bead tracking is not trivial to implement. To address this, we have developed FM-Track, a straightforward open source software written entirely in the Python language.

There are multiple examples of feature based particle tracking algorithms described in the literature. In this paper, we build on these methods and introduce a simple open source particle tracking software written entirely in the Python language. Important novel aspects of our code that are, to the author's knowledge, not found elsewhere in the literature, are (1) minor aspects of the tracking algorithm such as the procedure for checking different permutations of feature matches and (2) a multivariate adaptive regression spline (MARS) based translation correction function that is necessary to overcome potential microscope hardware limitations and unintended sample translation. A description of how FM-Track works is given in Section 2 and a minimum working example is presented in Section 3.

2. Software description:

Many open source particle tracking algorithms are designed for systems where there is high time resolution between images. Here, we focus on systems where there are only two images: one initial image and one final image. In this scenario, a successful algorithm must be able to accommodate large deformation between time frames, but computational efficiency is less critical given that the full algorithm will only have to run once per image set and will never run in real time. Specifically, the objective of this software is to go from two image stacks acquired with confocal microscopy (or an equivalent technique) of fluorescent beads and stained cells to a matched set of bead center positions. This process is broken down into three steps: pre-processing, tracking, and post-processing and the software is written to easily accommodate different sets of input parameters. To operate the software, the user will make changes to parameters and file names in "`input_info.py`" and, if necessary, make changes to the workflow in "`run_tracking_all_steps.py`". The file "`run_tracking_all_steps.py`" is set up such that the user can run the algorithm on a set of images by calling the script once.

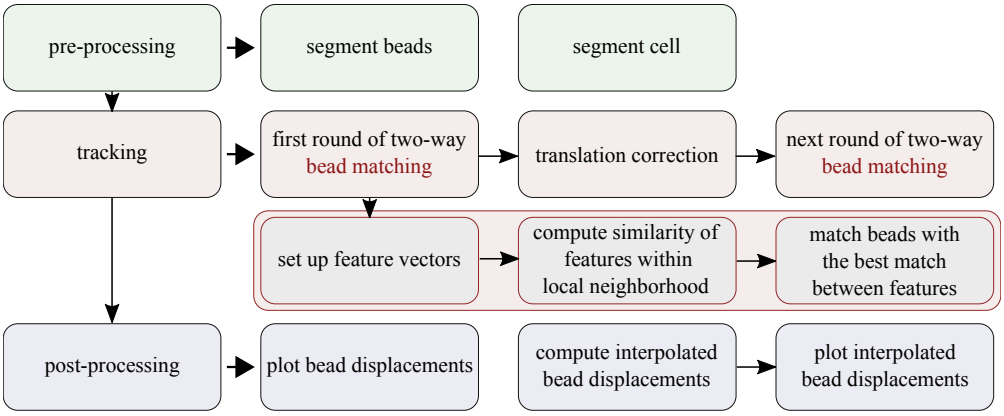

Figure 1: Basic flow chart of FM-Track.

2.1 Software architecture:

Work flow in FM-Track is broken down into three types of functions: pre-processing, tracking, and post-processing. These functions are defined in three separate files and are all called from the main script "`run_all_track.py`". User-specified input parameters and paths to image files are identified in a separate python file "`input_info.py`". The flow chart of FM-Track is illustrated in Fig. 1.

2.2 Software functionalities:

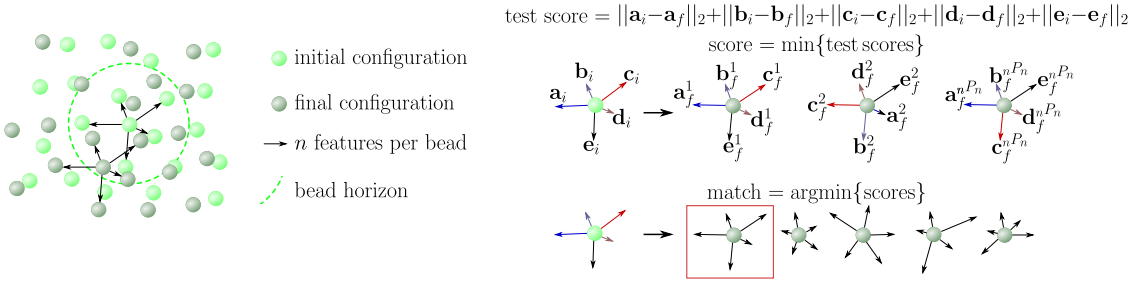

Figure 2: Schematic of the algorithm for feature based bead matching. The ${}_nP_n$ permutations of features for each bead within a defined horizon are compared, and the matched bead is the bead with the best score. Clashes are resolved by assigning the match to the pair with the better score. Two-way tracking means that the match must hold when the algorithm is run from initial configuration to final configuration and from the final configuration to the initial configuration.

The pre-processing file contains three functions. First, a function to import the three dimensional image stacks taken with a confocal microscope in the correct order as a `numpy` array. Second, a function to identify the coordinates of the center of every bead in a given image. Third, a function to identify the approximate volume, center, surface mesh, and surface mesh normals of a single cell in a given image. If there are multiple cells in the image, the function in its current state will only return the largest cell by volume.

The tracking file contains a main function "`track_main_call`" to perform bead tracking given a specified first and second set of input images. If tracking type 1 is specified, the tracking function will run the tracking algorithm once. If tracking type 2 is specified, the tracking function will run the tracking algorithm, run a translation correction algorithm, and then run the tracking function again. The translation correction algorithm will correct for any translation that can be described as a function of $z-$position. A general outline of how the tracking function works is illustrated in Fig. 2. Once the tracking step is complete,

the initial coordinates and displacement vector are recorded for every bead that was successfully matched. The tracking file also contains a similar function called `track_no_cell()` that is suitable for running the algorithm when there is no cell present.

The post-processing functions are included for easy visualization of the bead tracking results. Several different types of plots of the raw bead displacements are available, as illustrated in Fig. 3. Furthermore, the post processing step also includes a function to interpolate the raw bead displacements using Gaussian Process Regression (GPR) "`create_GP_model`" and a function to produce filled color plots of the interpolations "`plot_gp_model`".

3. Illustrative example:

To illustrate the functionality of FM-Track, we present an example of a porcine Aortic Valve Interstitial Cell (AVIC) embedded in a PEG hydrogel imaged under normal conditions and imaged after it is treated with Cytochalasin-D, a chemical that inhibits actin polymerization and therefore shuts down the contractile machinery of the cell. By comparing the treated cell to the normal (untreated) cell, we are able visualize how the cell actively deforms the surrounding gel. A brief overview of the experimental protocol used to generate these images is included in Appendix 1. In Appendix 2, we also included two additional examples with synthetic data derived from a finite element model of a block undergoing uniaxial extension. The synthetic data is useful for evaluating the performance of our tracking algorithm. On synthetic data, our tracking algorithm performs quite well on the relevant bead densities up to realistic expected levels of deformation.

Prior to running the code, the following changes must be made to the file `input_info.py`:

- Specify the path to all cell files by updating the function "`get_filenames_cell()`" (the images must be sequentially numbered, with 4 digits padded by zeros)
- Specify the path to all bead files by updating the function "`get_filenames_beads()`"
- Specify the unique filename that segmented cell and bead files will be saved as by updating the function "`get_savenames()`"
- Specify the tracking pairs by updating the function "`get_tracking_pairs()`" (these must match the unique filenames specified in "`get_savenames()`")
- If required, additional updates can be made to: the specified fluorescent color channels in the RGB input images "`get_color_channels()`", the microscope field of view dimensions "`get_FOV_dims(type)`", the threshold for segmenting the cell image "`get_cell_thresh()`", the tracking parameters "`get_tracking_params()`", the translation correction parameters "`get_translation_correction_params()`", and the type(s) of figure to save post processed results as "`get_figtype()`".

Once the parameters are specified in `input_info.py`, we run the script `run_tracking_all_steps.py`. This file is shown in and the components are as follows:

- Pre-process: get bead centers "`get_bead_centers()`", get cell surface meshes "`get_cell_surface()`"
- Track: run the tracking step "`track_main_call()`", the result of running this step are text files that contain the x, y, and z coordinates of each bead in the first configuration, and text files that contain the displacement of each bead u, v, and w

```
import input_info as info
import post_process as post_process
import pre_process as pre_process
import tracking as track
import numpy as np
import os
##########################################################################################
# indicate which steps to run
##########################################################################################
run_pre_process = True
run_tracking = True
run_post_process = True
##########################################################################################
# get set up info from input files
##########################################################################################
filenames_beads, dirnames_beads = info.get_filenames_beads()
cell_channel, bead_channel = info.get_color_channels()
X_DIM, Y_DIM, Z_DIM = info.get_FOV_dims(1)
cell_threshold = info.get_cell_thresh()
filenames_cell, dirnames_cell  = info.get_filenames_cell()
savefnames_cell, savefnames_beads  = info.get_savenames()
tracking_pairs = info.get_tracking_pairs()
num_feat, num_nearest, buffer_cell, track_type  = info.get_tracking_params()
figtype_list, plot_type, run_GP, use_corrected_cell = info.get_postproc_info()
##########################################################################################
# pre-process
##########################################################################################
num_imgs = len(filenames_beads)
if run_pre_process:
  for kk in range(0,num_imgs):
    pre_process.get_bead_centers(dirnames_beads[kk],filenames_beads[kk],savefnames_beads[kk
    ],bead_channel, X_DIM, Y_DIM, Z_DIM)
    pre_process.get_cell_surface(dirnames_cell[kk],filenames_cell[kk],savefnames_cell[kk],
    cell_channel, X_DIM, Y_DIM, Z_DIM, cell_threshold)
##########################################################################################
# run tracking
##########################################################################################
num_tracking_pairs = len(tracking_pairs)
if run_tracking:
  for kk in range(0,num_tracking_pairs):
    closest_no_conflict = track.track_main_call(track_type,tracking_pairs[kk][0],
    tracking_pairs[kk][1], num_feat,num_nearest, buffer_cell)
##########################################################################################
# post-process
##########################################################################################
num_tracking_pairs = len(tracking_pairs)
if run_post_process:
  for kk in range(0,num_tracking_pairs):
    post_process.call_plot_main(plot_type,tracking_pairs[kk][0], tracking_pairs[kk][1],
    num_feat,X_DIM,Y_DIM,Z_DIM,figtype_list, use_corrected_cell)

  if run_GP:
    for kk in range(0,num_tracking_pairs):
      post_process.create_GP_model(tracking_pairs[kk][0], tracking_pairs[kk][1])
      post_process.plot_gp_model(tracking_pairs[kk][0], tracking_pairs[kk][1],X_DIM,Y_DIM,
    Z_DIM,figtype_list)
```

Listing 1: Main file.

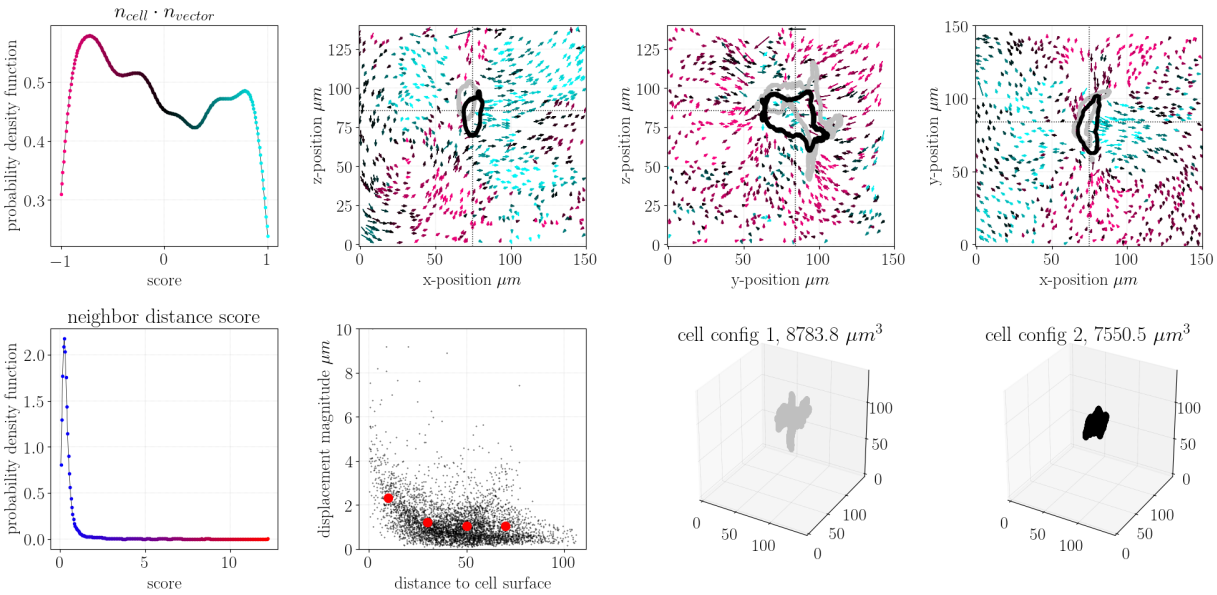

Figure 3: Summary of the results from FM-Track. Upper left: distribution of $n_{cell} \cdot n_{vector}$ where n_{cell} is the outward facing unit normal at the closet point on the cell surface and n_{vector} is the unit normal vector direction of bead motion; Upper right: projections of bead displacement through plane slices, where the gray outline is the cell in the initial configuration, the black outline is the cell in the final configuration, and bead displacements are color coded by direction compared to the cell surface; Lower left: comparison of bead displacements to their immediate neighbors (beads with high scores may be spurious); Lower center: bead displacement magnitude with respect to distance from cell surface; Lower right: three-dimensional plots of the cell in each configuration.

- Post-process: plot the raw data "`call_plot_main()`", generate the Gaussian Process Regression (GPR) model to interpolate bead displacements "`create_GP_model()`", plot the GPR model "`plot_gp_model()`" (note: generating the Gaussian Process model may be slow and the saved model may take up a large amount of file space)

4. Impact and conclusions:

The main benefit of FM-Track is that it provides an open source and simple to implement feature based bead tracking algorithm in the Python framework. Users have the option to implement the code directly as specified in the working example, use it with altered input parameters, or build on the functions and framework presented here in a manner that is entirely specific to their own needs. Important novel aspects of our code that are, to the author's knowledge, not found elsewhere in the literature, are (1) minor aspects of the tracking algorithm (2) the MARS based translation correction function, and (3) an open source Python based framework. Looking forward, we anticipate that FM-Track will enable research in Traction Force Microscopy and related techniques.

Acknowledgments:
Funding

Appendix 1 Experimental protocol used to generate the data used in Section 3

Appendix 1.1 Sample preparation

Aortic heart valve leaflets were dissected from porcine hearts obtained from a local abattoir. Aortic valve interstitial cells (AVICs) were extracted from the leaflets via previously published methods [1]. AVICs

were seeded at a density of $500,000$ cells/mL along with 0.5 μm yellow-green fluorescent polystyrene microbeads (Polysciences) at a density of 4.14×109 (TODO ALEX WHAT DOES THIS MEAN) beads/ml within peptide-modified, poly (ethylene glycol) hydrogels comprised of MMP degradable peptide crosslinks (Bachem), CRGDS adhesive peptides (Bachem), 8-arm 40 kDa PEG-norbornene (Jenkem), lithium phenyl-2,4,6 – trimethylbenzophosphinate (LAP) photoinitiator (Sigma-aldrich), and PBS (Sigma-aldrich). The solution was pipetted into a petri dish with a 12 mm glass insert (Fisher Scientific) and polymerized underneath UV light (365 nm, 2.5 mW/cm^2) for 3 minutes. After that, the hydrogel construct was incubated at standard conditions in growth media for 72 hours before any experimentation was performed.

Appendix 1.2 3D Traction Force Microscopy Experiment

Image stacks ($150 \times 150 \times 120$ μm at 0.8 um z-spacing) of a single VIC and the immediately surrounding fluorescent microbeads were captured using a Zeiss LSM 710 inverted confocal microscope with a 63x, 1.4 numerical aperture water-immersion objective lens (Zeiss). Two total image stacks were taken for every cell under normal and non-contractile conditions. For the normal case, the sample was incubated for 40 minutes within Tryode's Salt Solution (TSS, Sigma Aldrich) before the first image stack was acquired. After that, a non-contractile state was induced by adding a stock solution of Cytochalsin D (dissolved in DMSO, Sigma Aldrich) to achieve a final concentration of 4 μM. A second image stack of the same FOV was acquired after 40 minutes of incubation.

Appendix 2: Testing FM-Track performance on synthetic data

Appendix 2.1 Note on synthetic data

Synthetic data is generated by randomly identifying points in the reference configuration of finite element simulations. The initial fiducial marker locations are the randomly identified points in the reference configuration, and the final fiducial marker locations are the randomly identified points in the current configuration after some boundary conditions or loading has been applied in the finite element setting. Notable, we shuffle the order of the fiducial markers in the final configuration in our synthetic data set.

Appendix 2.1 Performance on synthetic data

Here we show the performance of our tracking algorithm on a synthetic data set. Fig. 4a shows the finite element simulation used to generate the synthetic data, and Fig. 4b shows the method for adjusting the synthetic data set to contain noise. In Fig. 4c, we show the performance of FM-Track at two different marker densities, and on a data set with progressively higher levels of random perturbation. Notably, FM-Track performs quite well on block stretch up to 1.10, and with small random perturbations, and when it fails it tends to fail by not matching beads rather than by reporting incorrect matches.

References

1 C. M. Johnson, M. N. Hanson, and S. C. Helgeson. Porcine cardiac valvular subendothelial cells in culture: cell isolation and growth characteristics. *Journal of molecular and cellular cardiology*, 19(12):1185–1193, 1987.

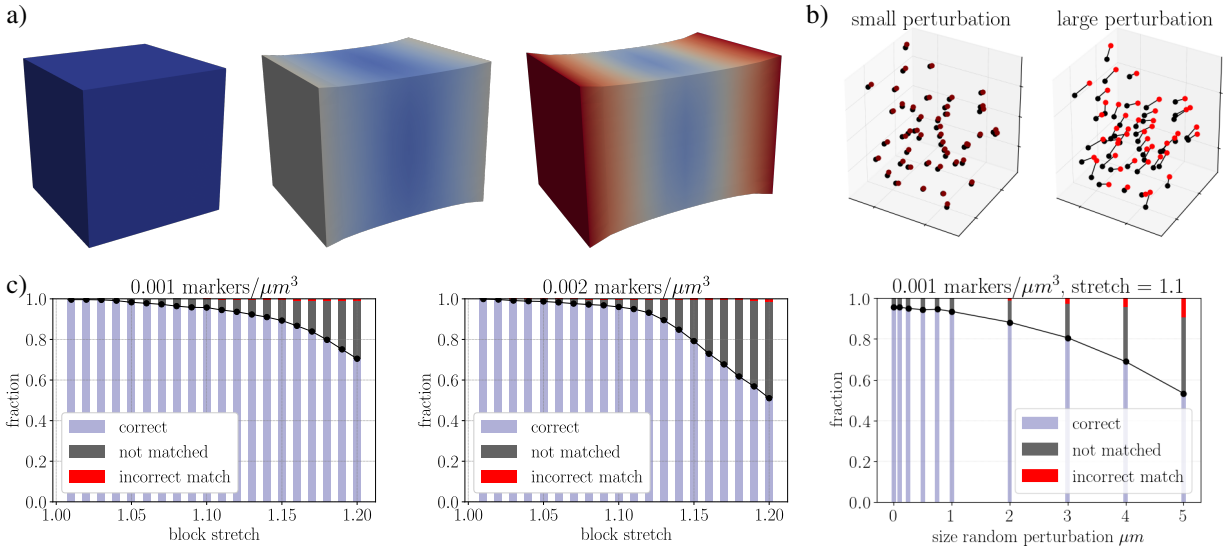

Figure 4: a) illustration of a finite element model of a $100\,\mu m \times 100\,\mu m \times 100\,\mu m$ block under uniaxial displacement controlled extension; b) illustration of random perturbations to the bead positions, where the size of the random perturbation indicates the maximum perturbation in a single direction; c) summary of FM-Track performance on this synthetic data set.