# SerialTrack: ScalE and Rotation Invariant Augmented Lagrangian Particle Tracking Code Manual (v1.0)

Jin Yang[1], Yue Yin[2], Alexander K. Landauer[3], Selda Buyuktozturk[1,4], Jing Zhang[1], Luke Summey[1], Alexander McGhee[1], Matt K. Fu[5], John O. Dabiri[5], Christian Franck[1,†]

[1] Department of Mechanical Engineering, University of Wisconsin-Madison, Madison, WI, USA
[2] Department of Mechanical Engineering, Carnegie Mellon University, Pittsburgh, PA, USA
[3] National Institute of Standards and Technology, Gaithersburg, MD, USA
[4] School of Engineering, Brown University, Providence, RI, USA
[5] Graduate Aerospace Laboratories, California Institute of Technology, Pasadena, CA, USA

Email: [†]cfranck@wisc.edu

Github page: https://github.com/FranckLab/SerialTrack

Last updated on July 15, 2022

# Contents

# 1 Introduction

This code manual is to help guide the user through running our particle tracking algorithm – ***ScalE and Rotation Invariant Augmented Lagrangian Particle Tracking (SerialTrack)*** – to measure full-field ***2D & 3D*** motions with potentially large deformations and rotations. SerialTrack takes advantage of both local (individual or several particle-scale motions) and global (entire field of view) particle tracking schemes. It iterative builds a scale and rotation invariant topology-based feature vector for each particle within a multi-scale tracking algorithm framework. Displacement and deformation gradients for each particle are estimated by matching reference and deformed particle locations to these features across frames. Global kinematic compatibility is enforced as a global augmented Lagrangian constraint on the collective particle motion field estimate to enhance the overall tracking performance.

An open-source MATLAB implementation is provided to execute our SerialTrack method and apply it to their research projects, either as-is or in a customized workflow. The SerialTrack code follows the workflow shown in Fig. 1. The overall particle tracking algorithm initialization requires input from the user regarding the spatial ***dimension***, ***particle rigidity***, and specific ***tracking mode*** desired for the particular application. For all algorithm configurations, the tracking process in itself includes three main procedures: (i) ***particle detection***, (ii) ***particle linking***, and (iii) ***post-processing***. Each of these procedures has a selection of user-defined parameters, and their usage is describe later in this manual. For more details of the SerialTrack method and a summary of other state-of-art particle tracking codes, please refer to our manuscript (full text can also be requested via the ResearchGate link [1] or the arXiv link [2]).

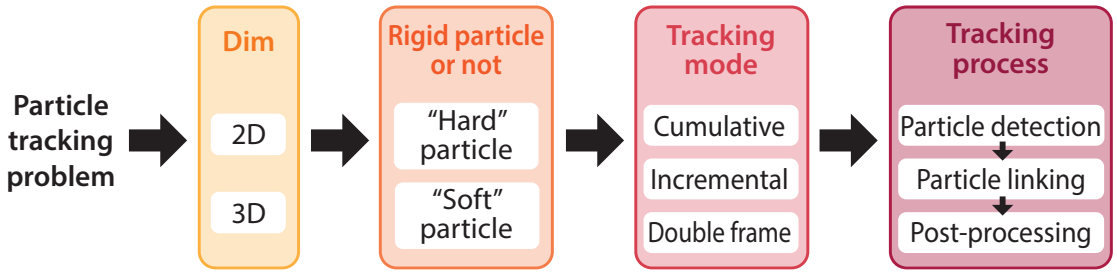


Figure 1: General SerialTrack initialization workflow. The particle tracking setup is first defined by the dimensionality, i.e., either as 2D (pixel image) or 3D (volumetric voxel image), of the images and motion field. In the second steps, hard vs. soft particle option specifies whether the tracker particles are expected to deform ("Soft") or will preserve their shape throughout the image sequence ("Hard"). In the third step, the tracking mode defines the scheme for selecting images from the experimental image sequence. Finally, fourth, the particle detection, linking, and post-processing strategies are specified based on the experimental configuration and desired output data.

Generally, particle tracking is one of many methods to reconstruct full-field motions or deformations in experiments. Besides tracking individual particles, there are also subset-based tracking methods including Particle Image Velocimetry (PIV), 2D Digital Image Correlation (DIC), 3D stereo

Digital Image Correlation (3D-DIC), and 3D Digital Volume Correlation (DVC) methods. A comparison of several examples of these methods to SerialTrack can be found in Table 2 of our main manuscript.

## 2 Code Installation

The SerialTrack code has been tested on MATLAB version R2021b and can be downloaded from our GitHub repository [3]. [ToCheck] on both Windows 10 and MacOS? operation systems. The Parallel Computing Toolbox is recommended (not required) to speed up the code. Required MATLAB toolboxes are summarized in Table 1, row C7, and include the Curve Fitting Toolbox, Image Processing Toolbox, Statistics and Machine Learning Toolbox, and Wavelet Toolbox.

To install the code, download and unzip the code folder and add the folder to the MATLAB working directory by right-clicking the folder and selecting “add to path” and “Selected Folders and Subfolders.” The SerialTrack code includes two subfolders called “SerialTrack2D” and “SerialTrack3D” to track 2D and 3D motions/deformations, respectively (see Fig. 2).

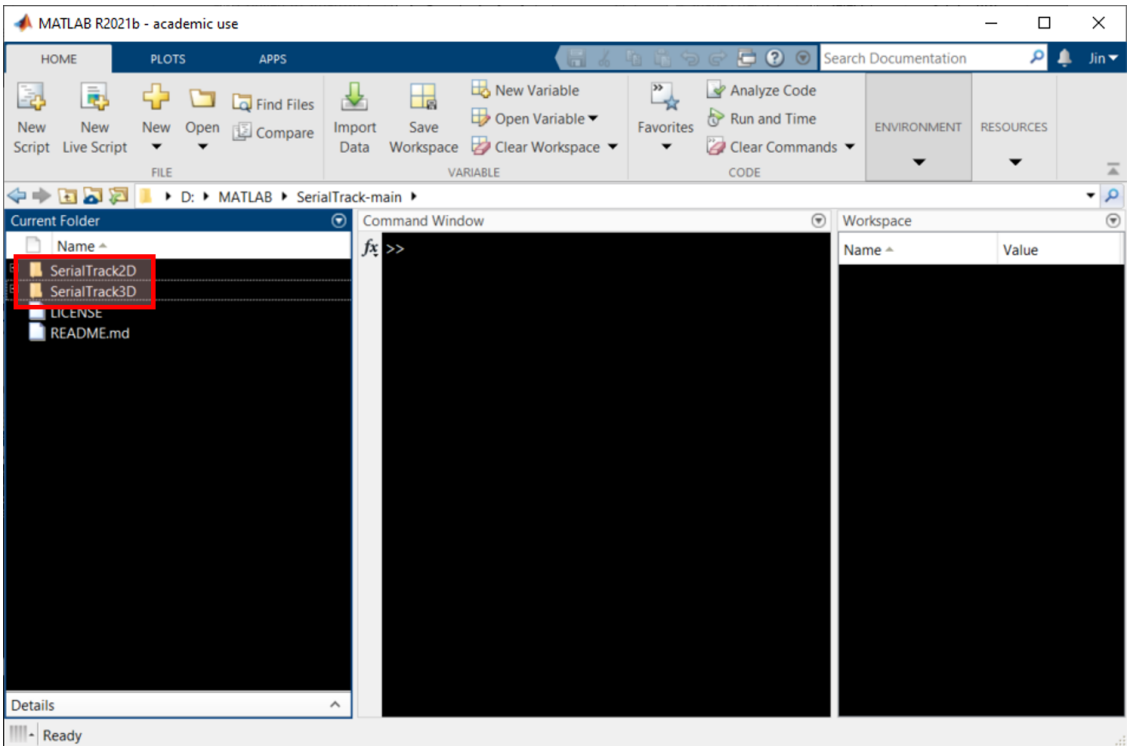


Figure 2: The main directory of the SerialTrack MATLAB code package in the MATLAB integrated development environment. The SerialTrack package includes two subfolders, “SerialTrack2D” and “SerialTrack3D”, to track 2D and 3D motions/deformations, respectively.

To execute the SerialTrack particle tracking algorithm, we provide several examples of main files in two subfolders (“SerialTrack2D” and “SerialTrack3D”) which can be executed without modification (see Fig. 3 yellow box). We also provide example data sets via the MINDS@UW open access

| Nr. | Code metadata description | Code release specification |
|---|---|---|
| C1 | Current code version | v1.0 |
| C2 | Permanent link to code/repository used for this code version | https://github.com/FranckLab/SerialTrack |
| C3 | Code Ocean compute capsule | Not applicable |
| C4 | Legal Code License | MIT license |
| C5 | Code versioning system used | git |
| C6 | Software code languages, tools, and services used | MATLAB[1] |
| C7 | Compilation requirements, operating environments & dependencies | MATLAB with the following toolboxes: Curve Fitting Toolbox, Image Processing Toolbox, Parallel Computing Toolbox, Statistics and Machine Learning Toolbox, Wavelet Toolbox |
| C8 | If available Link to developer documentation/manual | https://github.com/FranckLab/SerialTrack/manual.pdf |
| C9 | Support email for questions | cfranck@wisc.edu |

Table 1: Code metadata and release information

[1]Certain commercial equipment, software and/or materials are identified in this paper in order to adequately specify the experimental procedure. In no case does such identification imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the equipment and/or materials used are necessarily the best available for the purpose.

institutional data repository [4]. These data sets and other user-provided experimental data sets (at least two frames, see guidelines below for image recommendations) are stored in a subfolder, e.g., “imgFolder” as shown in Fig. 3.

To run the included demonstration examples, open the main run scripts whose file names have the format of “ `Example_main_*.m` ”, as shown by the yellow box in Fig. 3. Then the user can execute the entire script by clicking the “Run” button or pressing “F5” on the keyboard. Each mfile script can also be executed section by section by clicking the “EDITOR >> RUN >> ▶” button (see Fig. 4A) or pressing Control+Enter on the keyboard.

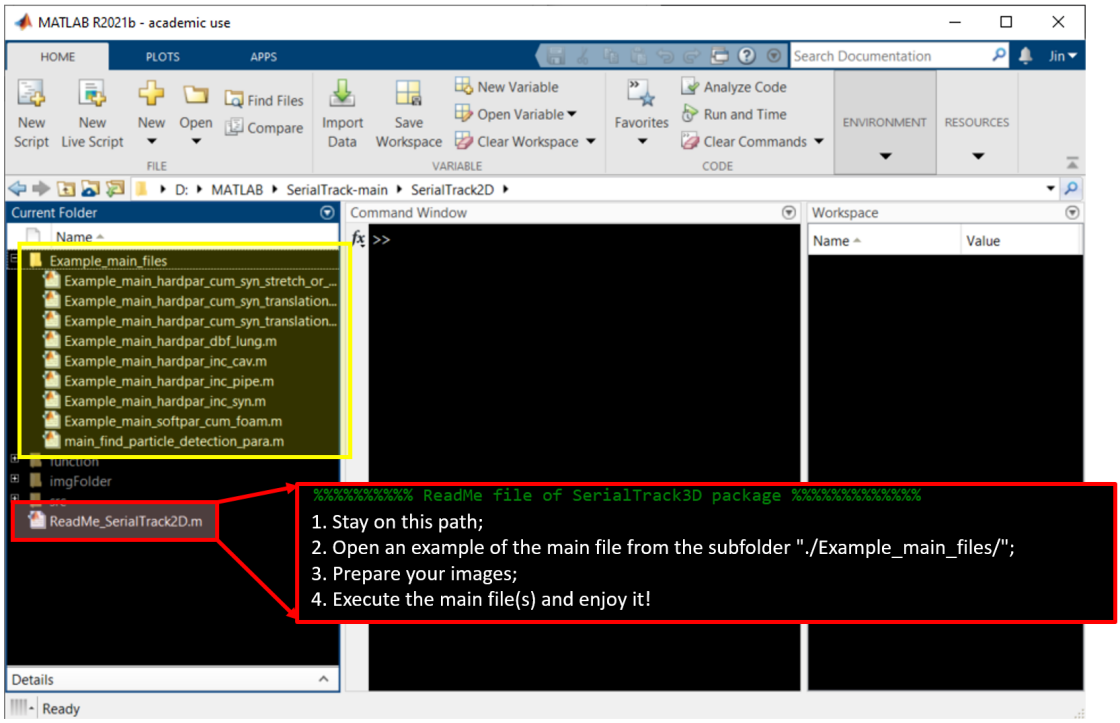


Figure 3: Example main files are stored in the subfolder “ `./Example_main_files/` ”

This manual showcases the SerialTrack code package on several typical 2D & 3D examples in Section 3 and Section 4.

# 3 SerialTrack 2D Examples

## 3.1 2D rigid body motions: synthetic translations and rotations

The first 2D example is tracking 2D rigid body motions including translations and rotations by executing the mfile “ `Example_main_hardpar_accum_syn_translation_or_rotation.m` ”, which is stored in subfolder “ `./SerialTrack2D/Example_main_files/` .”

At the beginning of the main mfile script, there is a file header (see Fig. 4B) to explain its primary goal:

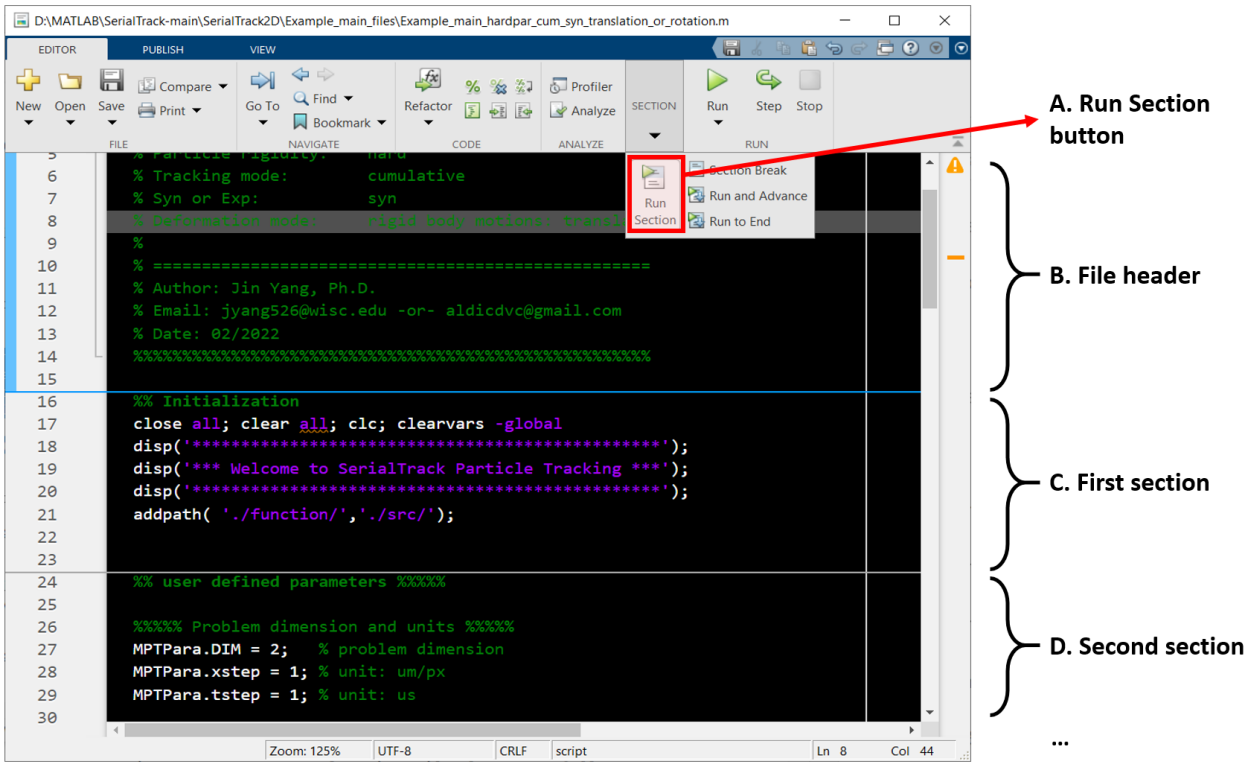


Figure 4: A screenshot of " `Example_main_hardpar_accum_syn_translation_or_rotation.m` " script. (A) The button to execute the selected code section. (B-D) File header, first, and second sections of one typical main script.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% SerialTrack execute main file
% ================================================
% Dimension:              2D
% Particle rigidity:      hard
% Tracking mode:          cumulative
% Syn or Exp:             syn
% Deformation mode:       rigid body motions: translations & rotations
%
% ================================================
% Author: Jin Yang, Ph.D.
% Email: jyang526@wisc.edu -or- aldicdvc@gmail.com
% Date: 02/2022
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

After the file header, the main file is organized into several sections (see Fig. 4C-D). Next, we will introduce these code sections in sub-Sections 3.1.1-3.1.8.

### 3.1.1 Initialization

The first section clears the MATLAB environment and adds the relevant subfolders (i.e., subfolders ' `./functions/` ', ' `./src/` ') to the MATLAB working directory, which reads as:

```
%% Initialization
close all; clear all; clc; clearvars -global
disp('************************************************');
disp('*** Welcome to SerialTrack Particle Tracking ***');
disp('************************************************');
addpath( './function/','./src/');
```

### 3.1.2 General user-defined parameters

Several user-defined parameters are needed for the algorithm to successfully track particles, and are defined within the " `MPTPara` " structure variable. The selection of these parameters is based on the problem at hand and the tracking mode selected in the workflow outlined in 1. Since this example is a 2D problem, we define `DIM` as `2` . In this example, both `MPTPara.xstep` (the micron-to-pixel ratio) and `MPTPara.tstep` (the time per image frame) are defined as `1` , which means that we choose "1 (μm/pixel" as our spatial unit and "1 (μs/frame)" as our temporal unit.

```
%% user-defined parameters %%%%%

%%%%% Problem dimension and units %%%%%
MPTPara.DIM = 2;   % problem dimension
MPTPara.xstep = 1; % unit: um/px
MPTPara.tstep = 1; % unit: us -or- 1/frame
```

We choose the cumulative tracking mode (i.e., `MPTPara.mode = 'accum'` ) in this example where all subsequent (deformed) frames are compared to the first (reference) frame. Because rigid body deformations in this example do not cause bead shape distortions, we define the particle rigidity as `MPTPara.parType = 'hard'` .

```
%%%%% Tracking mode %%%%%
MPTPara.mode = 'accum'; % {'inc': incremental mode;
                        %  'accum': cumulative mode;
                        %  'dbf': double frame}

%%%%% Particle rigidity %%%%%
MPTPara.parType = 'hard'; % {'hard': hard particle;
                          %  'soft': soft particle}
```

In this synthetic example, we track the entire image region. Therefore, there is no need to utilize a binary mask file to describe an irregular tracking region of interest. In the MATLAB code, we thus define `MaskFileLoadingMode` as `0` .

```
%%%%% Image binary mask file %%%%%
MaskFileLoadingMode = 0; % {0: No mask file
                         %  1: Load only one mask file for all frames;
                         %  2: Load one mask file for each single frame;
                         %  3: Load a MATLAB mat file for all frames;
```

### 3.1.3 Particle detection and localization parameters

Next, we define the particle detection and localization parameters. In the SerialTrack implementation, we leverage particle detection methods where the minimum size of the particles that can be effectively detected and localized is 3 pixels by 3 pixels. By default particles are detected by a Laplacian of Gaussian image filtering technique, followed by a Gaussian interpolation of the particle peak intensities to localize the bead centroid [5, 6]. In this example, the particle detection parameters are summarized as follows and their physical explanations are described on the right.

```
%%%%% Particle detection parameters %%%%%
%%%%% Bead parameters %%%%%
BeadPara.thres = 0.5;           % Intensity threshold for detecting
                                % particles
BeadPara.beadSize = 3;          % Estimated radius of a single particle
                                % [px]
BeadPara.minSize = 2;           % Minimum area of a single particle [px^2]
BeadPara.maxSize = 20;          % Maximum area of a single particle [px^2]
BeadPara.winSize = [5, 5];      % [not used for 2D]
BeadPara.dccd = [1,1];          % [not used for 2D]
BeadPara.abc = [1,1];           % [not used for 2D]
BeadPara.forloop = 1;           % [not used for 2D]
BeadPara.randNoise = 1e-7;      % Random background noise to add to avoid
                                % non-uniqueness of saturated (uniformly
                                % black or white) areas
BeadPara.PSF = [];              % PSF function; Example: PSF = fspecial(
                                % 'disk',BeadPara.beadSize-1); % Disk blur
BeadPara.color = 'white';       % Bead color: 'white' -or- 'black'
```

Figure 5 summarizes a typical result of the particle detection process. Fig. 5(a) shows all the detected particles in the reference image. Fig. 5(b) visualizes three important particle detection parameters where `beadSize` is the estimated particle radius in pixels, `minSize` and `maxSize` are the estimated lower and upper bounds of the particle area in square pixels. Since threshold, and minimum and maximum area are challenging to directly estimate and critical for performance these parameters have an additional selection process. For the first image, the user is prompted with a summary of the threshold result as a binary image, and can iteratively refine the threshold value to correctly segment the particles from the background via accepting or rejecting the binarization and re-entering a threshold guess. After a threshold parameter is accepted, the user is shown a histogram of detected particle sizes and is prompted to enter and minimum and maximum particle size. This histogram visualization of the particle size distribution helps avoid capturing lossy features (background noise, large agglomerations of particles, etc.) by giving a direct measure of the segmented population.

For applications where particles are asymmetrical, hard to distinguish from the background, or otherwise uniquely challenging users are strongly encouraged to either develop a custom segmentation routine or contact the authors for assistance.

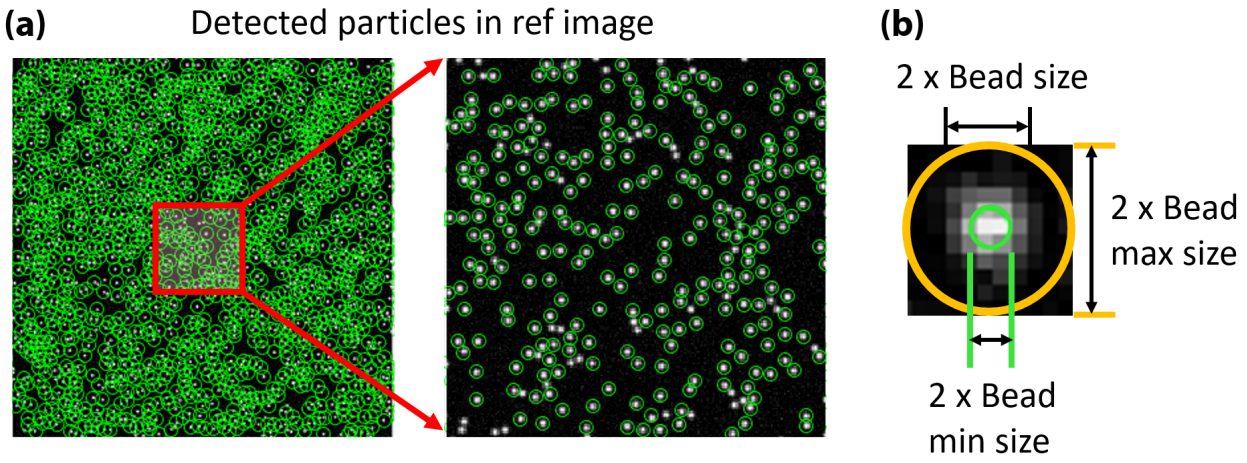


Figure 5: Particle detection results. (a) Detected particles in the reference image. (b) One typical detected particle with its detection parameters: beadSize, minSize, and maxSize.

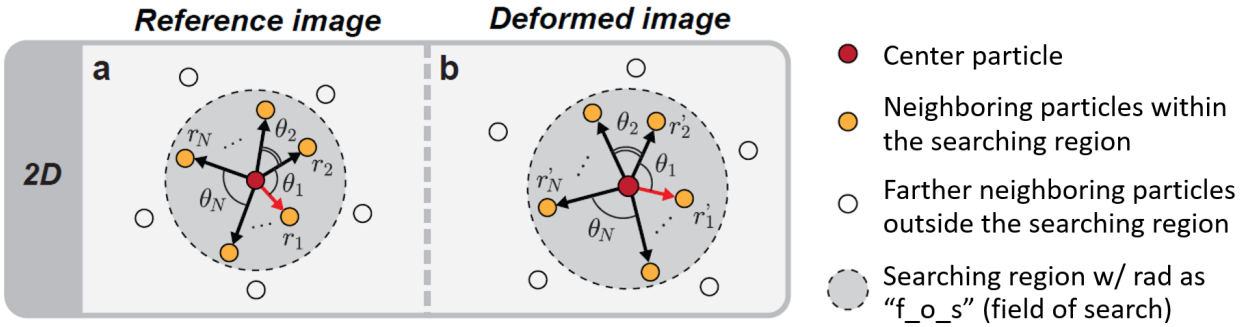


Figure 6: A diagram outlining the descriptor generation process. (a) The $k$ radii and angles to nearest neighbor particles within the search distance ( `f_o_s` ) for each particle. (b) The same computation is performed in the deformed image. By simultaneously minimizing the Euclidean distance for angular and distance mismatch, we achieve a fast linking procedure that is both scale and rotation invariant, and thus robust under most kinematically admissible deformations.

### 3.1.4 Particle linking parameters

After the particles have been successfully detected and localized, we need to define the particle linking parameters to automatically generate the ***topology-based scale and rotation-invariant particle descriptor*** for each particle detected in the reference and deformed configuration frames. Figure 6 provides an overview of the general linking procedure, outlining the feature descriptor generation process.

For each individual particle, the relative position between the k nearest neighboring particles and the selected particle is encoded into a particle descriptor consisting of two feature vectors. For the 2D case, an angle-based feature is defined as an array of polar angles between each of the k neighboring particles, i.e., $[\theta_1, \theta_2, \cdots, \theta_k]^T$. An array of interparticle Euclidean distances is also constructed as a distance-based feature, i.e., $[r_1, r_2, \cdots, r_k]^T$, where distances are normalized by

the first nearest neighbor particle distance.

If both `MPTPara.n_neighborsMax` and `MPTPara.n_neighborsMin` equal `1` , SerialTrack's local tracking approach is identical to the nearest neighbor search method. Alternatively, if both `MPTPara.n_neighborsMax` and `MPTPara.n_neighborsMin` equal the same integer number greater than one, SerialTrack's local tracking approach will be the same result as our previous topology-based particle tracking (TPT) method [7, 8]. If `MPTPara.n_neighborsMax` is greater than one and `MPTPara.n_neighborsMin` equals one, SerialTrack will start from the TPT method and converge to the nearest neighbor search method by iteratively decrementing the `MPTPara.n_neighborsMax` parameter. During this process, the deformed images are also iteratively warped based on the transiently tracked deformation fields.

By simultaneously minimizing the Euclidean distance for both the angular and distance mismatch, we achieve a fast linking procedure that is both scale and rotation invariant, and thus robust under most kinematically admissible deformations. More details of this linking procedure can be found in our original manuscript **Section 2.3**. The particle linking parameters and their descriptions are summarized below.

```
%% SerialTrack particle tracking

%%%%% Multiple particle tracking (MPT) Parameter %%%%%
MPTPara.f_o_s = Inf;              % Size of search [px] field: max(|u|,|v|)
MPTPara.n_neighborsMax = 25;      % Max # of neighboring particles
MPTPara.n_neighborsMin = 1;       % Min # of neighboring particles
MPTPara.locSolver = 1;            % Local solver:
                                  % 1-topology-based feature;
                                  % 2-histogram-based feature first and
                                  %   then topology-based feature;
MPTPara.gbSolver = 3;             % Global step solver:
                                  % 1-moving least square fitting;
                                  % 2-global regularization;
                                  % 3-ADMM iterations
MPTPara.smoothness = 1e-2;        % Coefficient of regularization
MPTPara.outlrThres = 2;           % Threshold for removing outliers in TPT
MPTPara.maxIterNum = 20;          % Max ADMM iteration number
MPTPara.iterStopThres = 1e-2;     % ADMM iteration stopping threshold
MPTPara.strain_n_neighbors = 20; % # of neighboring particles used in
                                  % strain gauge
MPTPara.strain_f_o_s = 60;        % Size of virtual strain gauge [px]
MPTPara.usePrevResults = 0;       % Whether use previous results or not:
                                  % 0-no; 1-yes;
MPTPara.distMissing = 2;          % Distance threshold to check whether
                                  % particle has a match or not [px]
```

### 3.1.5 Merging particle trajectory segments

This section of code merges the tracked particle trajectory segments, and is typically used in the incremental tracking mode. The key idea is that tracked, short segments can be further connected if they can be approximated by the same smooth, piecewise cubic spline interpolation/extrapolation

function. The explanations of these parameters are listed on the right. More details can be found in Section 3.3.4.

```
%%%% Postprocessing: merge trajectory segments %%%%%
distThres = 1;            % distance threshold to connect split
                          % trajectory segments [px]
extrapMethod = 'pchip';   % extrapolation scheme to connect split
                          % trajectory segments
                          % suggestion: 'nearest' for Brownian motion
minTrajSegLength = 0;     % the minimum length of trajectory segment that
                          % will be extrapolate [px]
maxGapTrajSeqLength = 0; % the max frame # gap between connected
                          % trajectory segments
```

### 3.1.6 Executing the SerialTrack particle tracking code

Finally, the SerialTrack particle tracking code will run the selected tracking mode including '***inc***': incremental tracking mode; '***accum***': cumulative tracking mode; '***dbf***': double frame tracking mode. We use the *cumulative* tracking mode in this example. *Incremental* and *double frame* tracking modes will be demonstrated later in Section 3.3 and Section 3.4, respectively.

```
%%%%% Execute SerialTrack particle tracking %%%%%
if strcmp(MPTPara.mode,'inc')==1
    run_Serial_MPT_2D_hardpar_inc;
elseif strcmp(MPTPara.mode,'accum')==1
    run_Serial_MPT_2D_hardpar_accum;
elseif strcmp(MPTPara.mode,'dbf')==1
    run_Serial_MPT_2D_hardpar_dbf;
end
```

### 3.1.7 Code execution results

After editing the parameters to reflect the current experiment, when executing the entire mfile script, " `Example_main_hardpar_accum_syn_translation_or_rotation.m` ", the user will be asked to load both the reference and deformed images. Here we load images by manually selecting the image folder, as shown in Fig. 7.

```
***********************************************
*** Welcome to SerialTrack Particle Tracking ***
***********************************************
***********************************************
Dimention: 2
Tracking mode: accum
Particle type: hard
***********************************************

Choose method to load images:
     0: Select images folder;
     1: Use prefix of image names;
     2: Manually select images.
Input here: 0
```

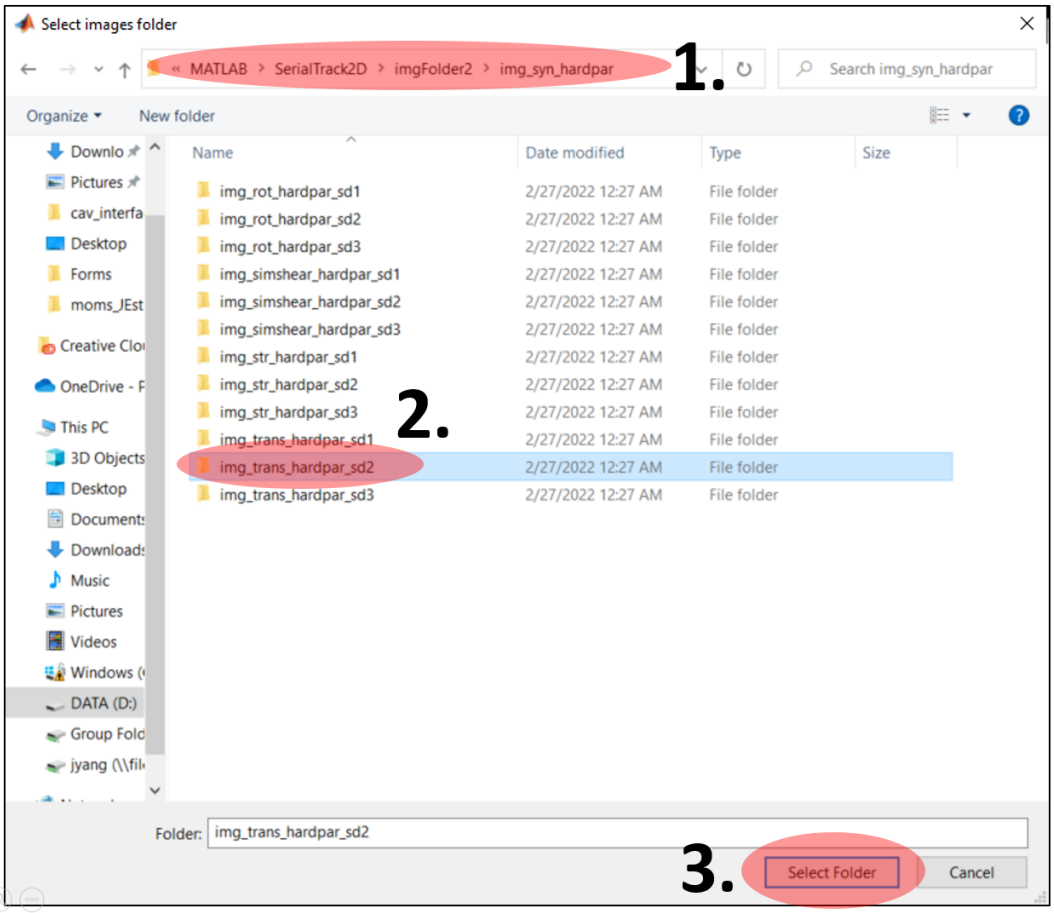


Figure 7: Images can be loaded by manually selecting an image folder where all frames are stored. Steps 1-2: To select the image folder with the stored images; Step 3: Click the "Select Folder" button to finish this step.

The code requires the user to define the region of interest within the reference image. Here we assume both reference and all deformed images to be of the same image size. The user can define a rectangular **region of interest (ROI)** by manually clicking a top-left point and a bottom-right corner point (see Fig. 8). If a top-left/bottom-right corner point is clicked out of the image, the code will update it as the nearest point on the edge. The MATLAB command prompt will also report the coordinates of the clicked corner points:

```
--- Define ROI corner points at the top-left and bottom-right ---
Coordinates of top-left corner point are (-34.000,-12.500)
Coordinates of bottom-right corner point are (537.000,542.500)
%%%%%% Load images: Done! %%%%%%
```

After defining the region of interest on the reference image, SerialTrack will start detecting and tracking particles in subsequent frames. The MATLAB command prompt will display the tracking results as shown below:

```
Detected particle # in ref image: 1538
%%%%%% Detect particles: Done! %%%%%%
```

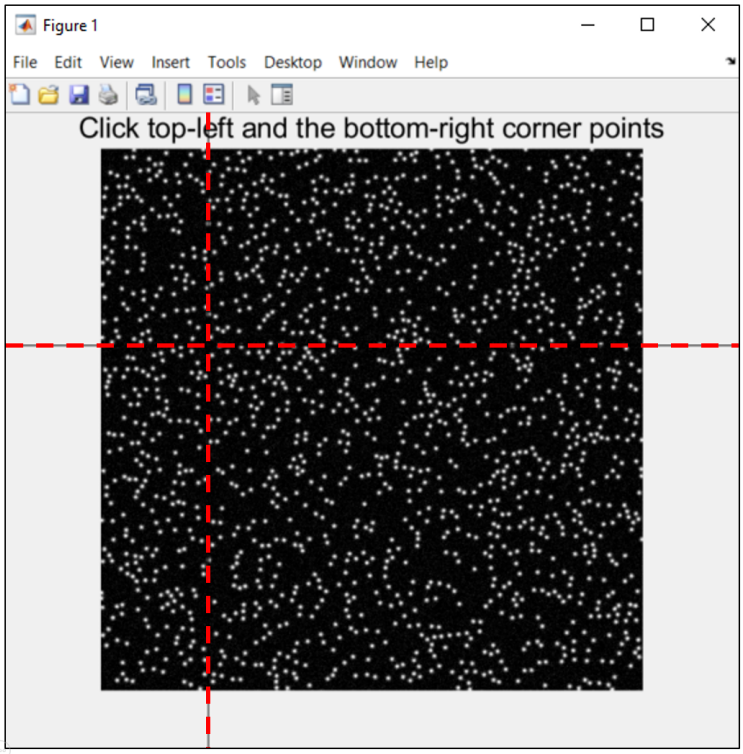


Figure 8: A rectangular region of interest (ROI) can be defined by manually clicking a top-left point and a bottom-right corner point

```
====== Frame #2 ======
Detected particle # in def image: 1538
------ Iter #1 ------
Tracking ratio: 1182/1538 = 0.76853
Disp update norm: 0.12613
------ Iter #2 ------
Tracking ratio: 1342/1538 = 0.87256
Disp update norm: 0.051094
------ Iter #3 ------
Tracking ratio: 1389/1538 = 0.90312
Disp update norm: 0.043043
------ Iter #4 ------
Tracking ratio: 1416/1538 = 0.92068
Disp update norm: 0.037033
------ Iter #5 ------
Tracking ratio: 1435/1538 = 0.93303
Disp update norm: 0.025455
------ Iter #6 ------
Tracking ratio: 1450/1538 = 0.94278
Disp update norm: 0.024545
------ Iter #7 ------
Tracking ratio: 1536/1536 = 1
```

```
Disp update norm: 0.028372
------ Iter #8 ------
Tracking ratio: 1536/1536 = 1
Disp update norm: 0.011581
----- Converged! ------
%%%%%% SerialTrack: Done! %%%%%%
Elapsed time is 1.098232 seconds.

====== Frame #3 ======
...
(Results of other frames can be found by executing the code.)
...

%%%%% Calculate incremental tracking ratio %%%%%

%%%%% SerialTrack hard particle tracking: Done! %%%%%

%%%%% Plot tracked cumulative deformations %%%%%

%%%%% Plot tracked trajectories %%%%%

Press "Ctrl + C" and modify the codes below to plot interpolated
   displacements and strains on a uniform grid mesh
```

The tracking results of the rigid body translations (Frames #2-#41 have rigid horizontal translations of [0.1 : 0.1 : 4] pixels) and rigid body rotations (Frames #2-#19 have rotation angles of 10-180 degrees in 10 degree increments) are summarized in Fig. 9 and Fig. 10, respectively.

### 3.1.8 Post-processing

As with other single particle tracking-based algorithms, SerialTrack's sampling of the motion field is inherently based on the original particle positions, and thus present an irregularly sampled displacement field. Thus, we also provide scripts to interpolate the scattered results onto a regularly spaced grid.

The user needs to assign the frame in the `ImgSeqNum` variable to perform the interpolation.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
disp(['Press "Ctrl + C" and modify codes below to plot interpolated ' ...
    'displacements and strains onto a uniform grid mesh']);
disp(['Press "Enter" key to keep running the code']);
pause;

ImgSeqNum = 2; % TODO: assign a Frame #
```

For example, the results of the second frame (i.e., `ImgSeqNum = 2` ) in Fig. 10(a) are shown after interpolation onto a rectangular mesh in Fig. 11.

```
%%%%% Previously tracked displacement field %%%%%
resultDispCurr = resultDisp{ImgSeqNum-1};
resultDefGradCurr = resultDefGrad{ImgSeqNum-1};
```

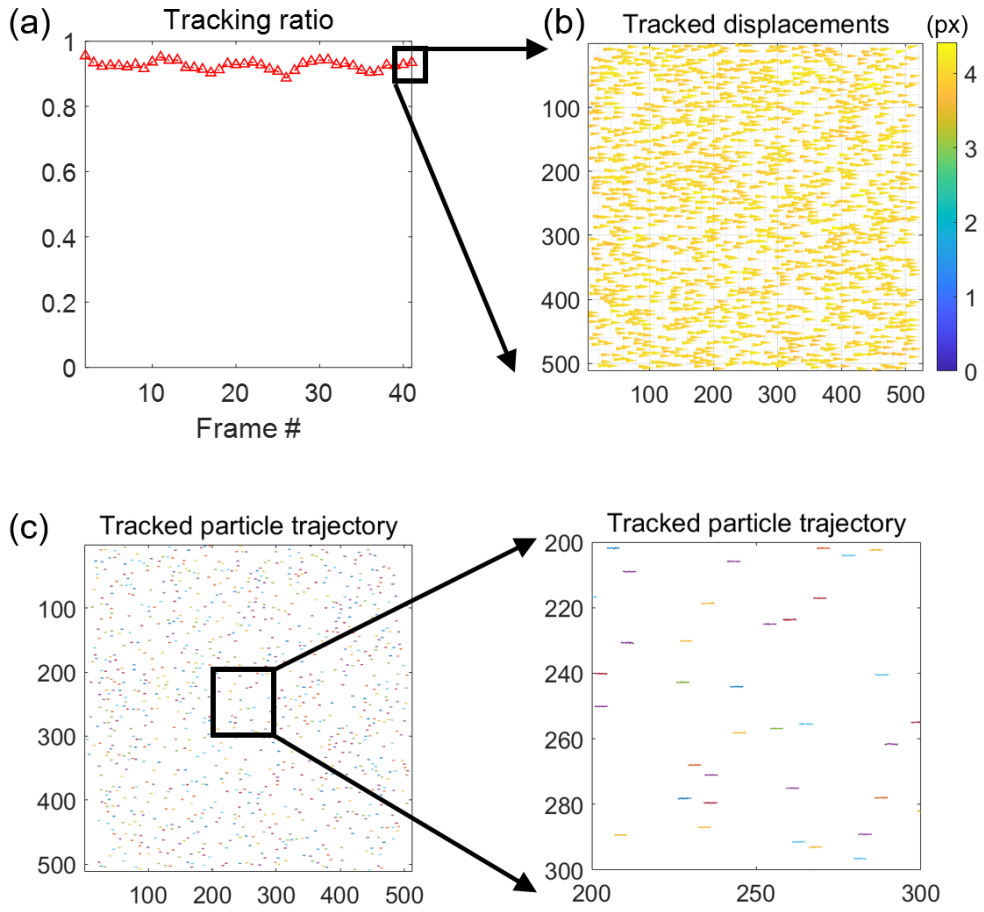


Figure 9: Tracked 2D rigid body translations. Frames #2-#41 have rigid horizontal translations of [0.1 : 0.1 : 4] pixels. (a) Final tracking ratios for subsequent frames. (b) Cone plot of the tracked displacement field of the 41st frame. (c) The trajectory plot of tracked particles, where a window of [200,300] pixels $\times$[200,300] pixels is shown enlarged on the right.

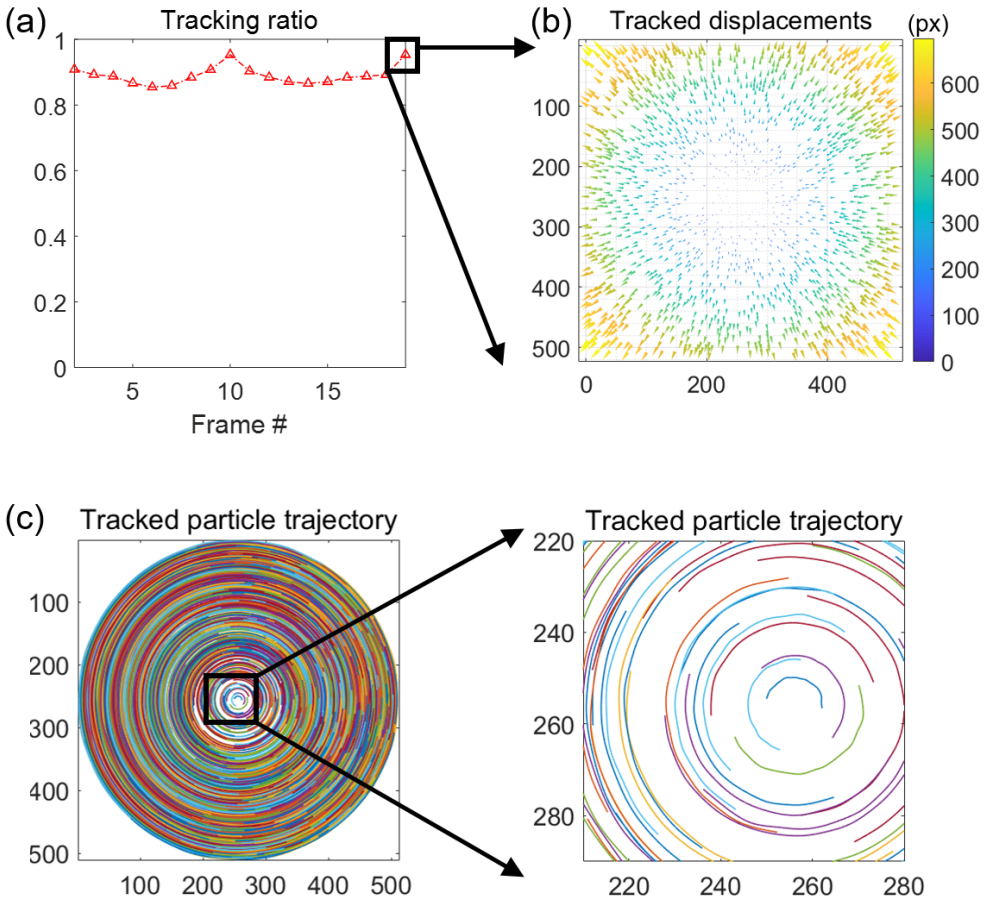


Figure 10: Tracked 2D rigid body rotations. Frames #2-#19 have rotation angles of 10-180 degrees in 10 degree increments. (a) Final tracking ratio for subsequent frames. (b) Cone plot of the tracked displacement field of the 18-th frame. (c) The trajectory plot of tracked particles, where a window of [210,280] pixel $\times$[220,290] pixel is shown enlarged on the right.

```
disp_A2B_parCoordB = resultDispCurr.disp_A2B_parCoordB;
parCoordB = resultDispCurr.parCoordB;

%%%%% Interpolate scatterred data to gridded data %%%%%
sxy = min([round(0.5*MPTPara.f_o_s),20])*[1,1]; % Step size for griddata [
    px]
smoothness = 1e-3; % Smoothness for regularization; "smoothness=0" means
                   % no regularization

[x_Grid_refB,y_Grid_refB,u_Grid_refB]=funScatter2Grid2D(parCoordB(:,1),
    parCoordB(:,2),disp_A2B_parCoordB(:,1),sxy,smoothness);
[~,~,v_Grid_refB]=funScatter2Grid2D(parCoordB(:,1),parCoordB(:,2),
    disp_A2B_parCoordB(:,2),sxy,smoothness);

% Apply ROI image mask
[u_Grid_refB, v_Grid_refB] = funRmROIOutside(x_Grid_refB,y_Grid_refB,
                             MPTPara.ImgRefMask,u_Grid_refB,v_Grid_refB);

% Build a displacement vector
uv_Grid_refB_Vector=[u_Grid_refB(:),v_Grid_refB(:)]';
uv_Grid_refB_Vector=uv_Grid_refB_Vector(:);

% Calculate deformation gradient
D_Grid = funDerivativeOp(size(x_Grid_refB,1),size(x_Grid_refB,2), ...
                         mean(sxy)); % Central finite difference operator
F_Grid_refB_Vector=D_Grid*uv_Grid_refB_Vector; % {F}={D}{U}

% Change "uv_Grid_refB_Vector" and "F_Grid_refB_Vector" in physical units,
    e.g., um or mm
uv_Grid_refB_Vector_PhysWorld = [u_Grid_refB(:)*xstep,v_Grid_refB(:)*ystep
                                 ]';
uv_Grid_refB_Vector_PhysWorld = uv_Grid_refB_Vector_PhysWorld(:);

F_Grid_refB_Vector_PhysWorld = [F_Grid_refB_Vector(1:4:end),
                                F_Grid_refB_Vector(2:4:end)*ystep/xstep,
                                F_Grid_refB_Vector(3:4:end)*xstep/ystep,
                                F_Grid_refB_Vector(4:4:end)]';
F_Grid_refB_Vector_PhysWorld = F_Grid_refB_Vector_PhysWorld(:);

%%%%% Cone plot grid data: displecement %%%%%
figure, plotCone2(x_Grid_refB*xstep,y_Grid_refB*ystep, ...
                  u_Grid_refB*xstep,v_Grid_refB*ystep );
set(gca,'fontsize',18); view(2); box on; axis equal;
axis tight; set(gca,'YDir','reverse');
title('Tracked displacement','fontweight','normal');
axis([xstep*MPTPara.gridxyROIRange.gridx(1), ...
      xstep*MPTPara.gridxyROIRange.gridx(2), ...
      ystep*MPTPara.gridxyROIRange.gridy(1), ...
      ystep*MPTPara.gridxyROIRange.gridy(2) ]);

%%%%% Generate an FE-mesh %%%%%
```

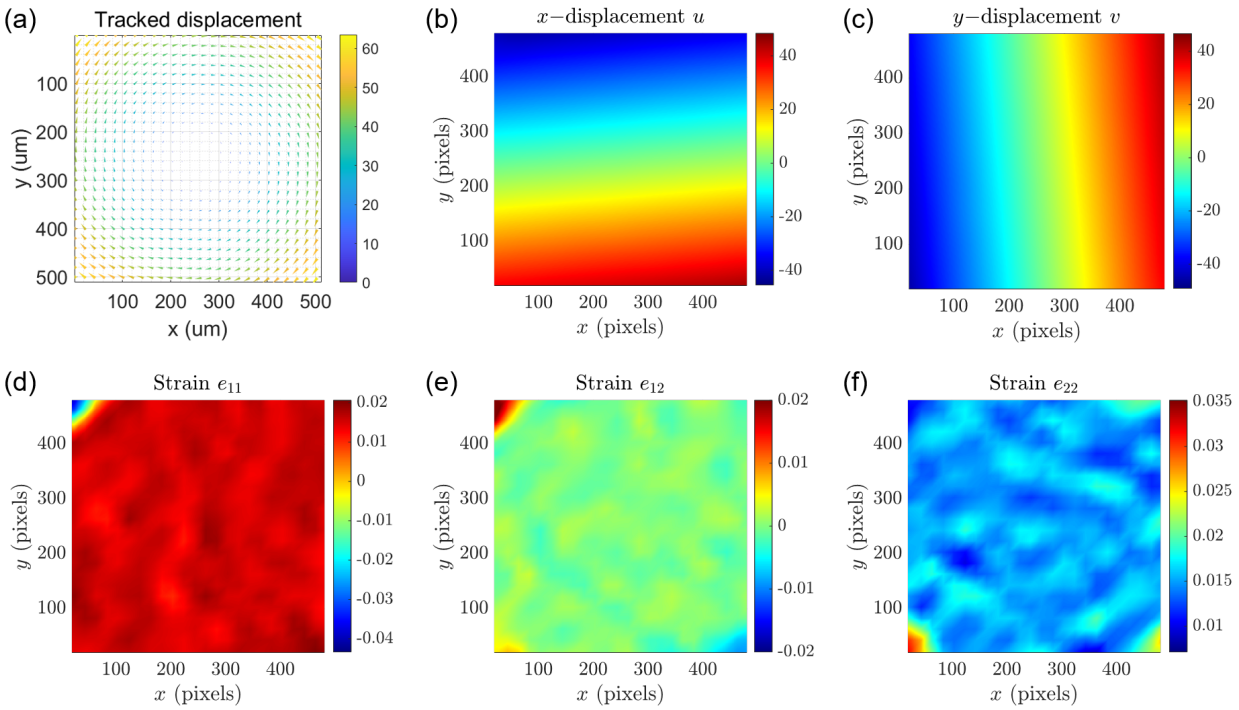


Figure 11: Results of the second frame ( `ImgSeqNum = 2` ) in Fig. 10(a) as interpolated onto a regularly spaced (structured) mesh. (a) Cone plot of the interpolated displacement field. (b-c) Interpolated x and y displacement components. (d-f) Interpolated $e_{xx}$, $e_{xy}$, and $e_{yy}$ strain fields.

```
[coordinatesFEM_refB,elementsFEM_refB] = funMeshSetUp(x_Grid_refB*xstep,
                                                      y_Grid_refB*ystep);

%%%%% Cone plot grid data: displacement %%%%%
Plotdisp_show(uv_Grid_refB_Vector_PhysWorld, coordinatesFEM_refB, ...
              elementsFEM_refB,[],'NoEdgeColor');

%%%%% Cone plot grid data: infinitesimal strain %%%%%
Plotstrain_show(F_Grid_refB_Vector_PhysWorld, coordinatesFEM_refB, ...
                elementsFEM_refB,[],'NoEdgeColor',xstep,tstep);
```

## 3.2 2D non-rigid body motion: synthetic uniaxial stretch and simple shear

Similarly to Section 3.1, another example of the 2D SerialTrack code is to track non-rigid body motions including uniaxial stretch (stretch ratios are from 1-3 in increments of 0.05) and simple shear deformations (shear angle $\gamma$ from 0° to 45° in tan($\gamma$) increments of 0.05). The user can execute the mfile " `Example_main_hardpar_accum_syn_stretch_or_shear.m` ", which is stored in subfolder " `./SerialTrack2D/Example_main_files/` ." Most steps are similar to Section 3.1. Here we summarize the tracked displacement fields of uniaxial stretch and simple shear in Fig. 12 and Fig. 13, respectively.

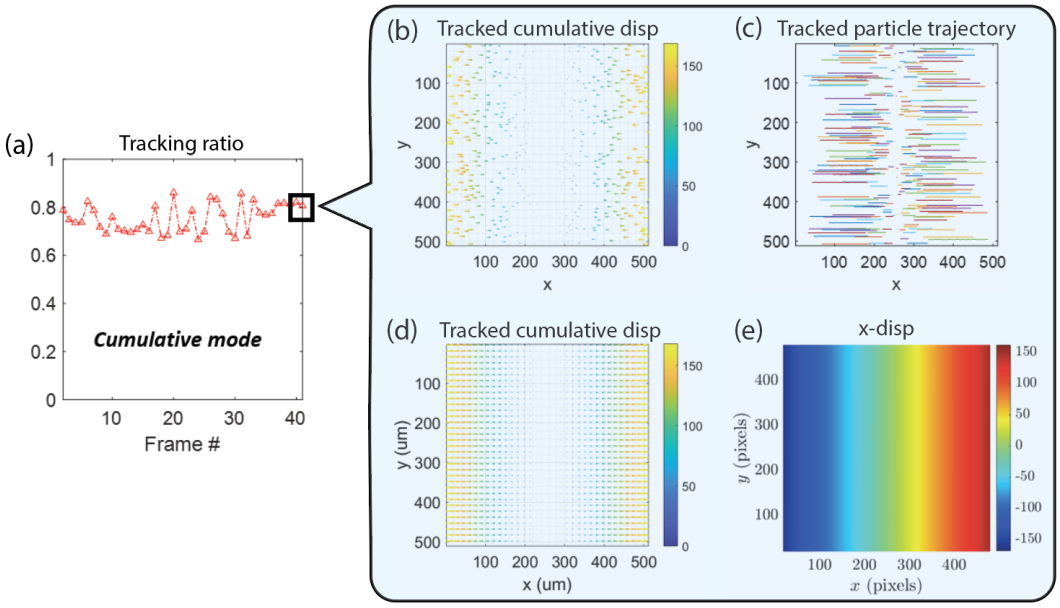


Figure 12: Tracked 2D uniaxial stretch deformations. Stretch ratios range from 1-3 in increments of 0.05. (a) Final tracking ratios. (b) Cone plot of the tracked cumulative displacement field. (c) Plot of tracked particle trajectories. (d) The interpolated tracked cumulative displacement on a regular, structured grid. (e) The interpolated tracked $x$-displacement field.

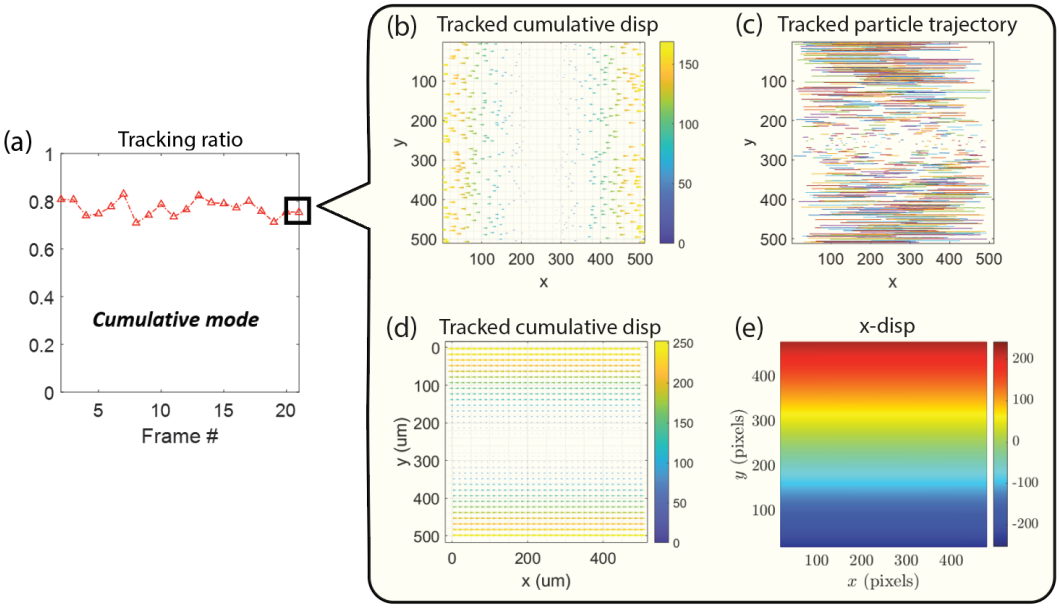


Figure 13: Tracked 2D simple shear deformations. Shear angle $\gamma$ ranges from 0° to 45° in $\tan(\gamma)$ increments of 0.05. (a) Final tracking ratios. (b) Cone plot of the tracked cumulative displacement field. (c) Plot of tracked particle trajectories. (d) The interpolated tracked cumulative displacement on a regular, structured grid. (e) The interpolated tracked $x$-displacement field.

### 3.3 2D complex geometry assisted with a mask file: flow through a bent pipe

In this example, we will show how to track a 2D motion field in a complex geometry via the use of a binary image mask. The data set featured here came from an experimental test of a pipe flow and can be obtained from [6]. To properly reconstruct a motion field within such a complex geometry, a prior ROI mat file or a binary mask file or a sequence of binary mask files needs to be prepared and saved on the MATLAB working directory (see Fig. 14). This mat file needs to include a saved binary matrix (or multiple binary mask files) of the same size as the original reference image. **In this binary matrix, "0" defines the background and "1" defines the data, or tracking, area of within the image. Only particles within the tracking area will be detected and tracked.** The utilized mask file is shown in Fig. 15(b).

In addition, after executing the code (" `Example_main_hardpar_inc_pipe.m` "), the user is requested to define the region of interest (ROI) by clicking a top-left and a right-bottom corner points (see Fig. 15(a)). **The final tracking area is the intersection of the manually defined ROI and the mask file.**

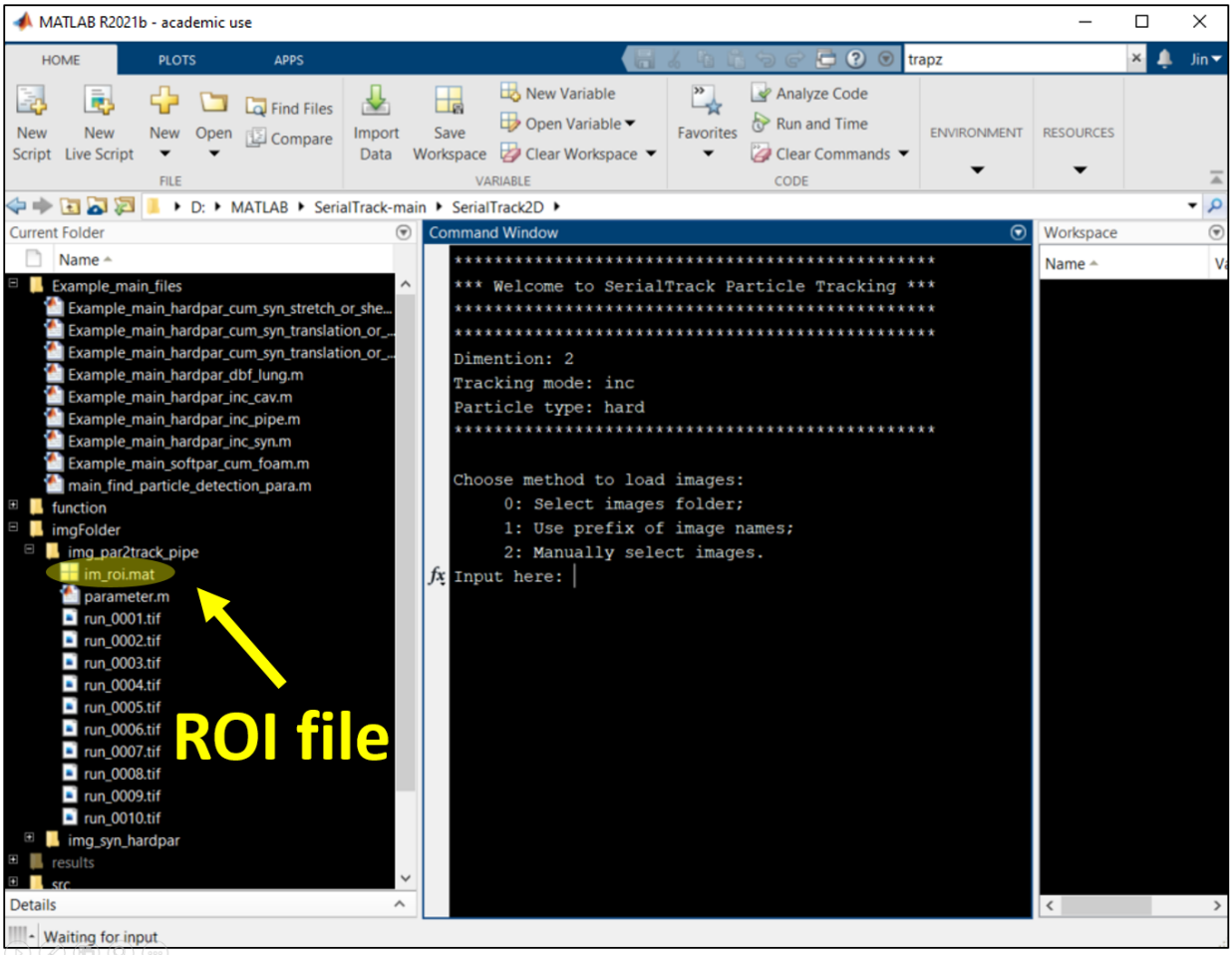


Figure 14: A prior ROI mat file needs to be prepared and stored to track motions in a geometrically complex configuration.

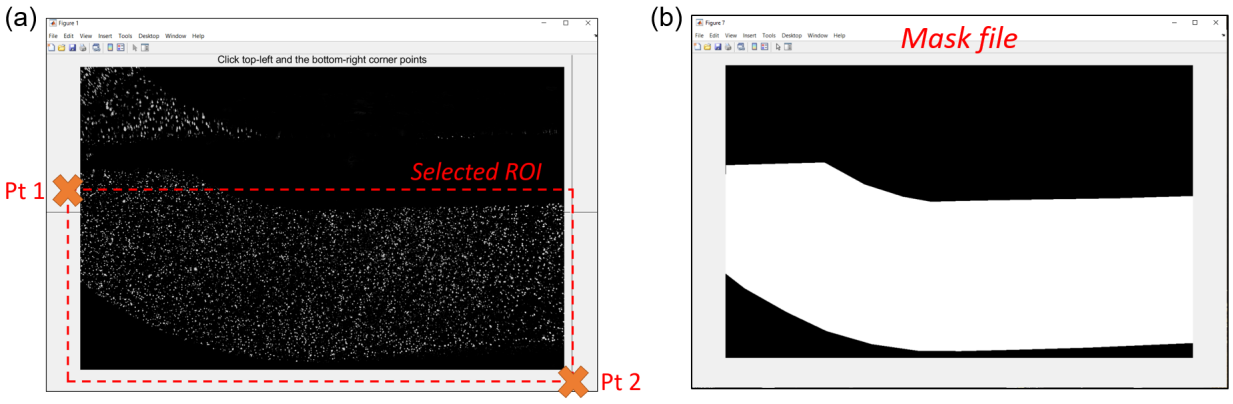


Figure 15: (a) A ROI can be defined by the user by manually clicking the top-left and bottom-right corner points. (b) The mask file used in this example.

### 3.3.1 User-defined parameters

The user can customize the particle detection and linking parameters as before. Here, we utilize real experimental spatial and temporal units.

```
%% User-defined parameters %%%%%

%%%%% Problem dimension and units %%%%%
MPTPara.DIM = 2;                % problem dimension
MPTPara.xstep = 0.075;          % unit: mm/px
MPTPara.tstep = 7.6923e-4;      % unit: s
```

We choose the incremental tracking mode (i.e., `MPTPara.mode = 'inc'` ) since large deformations can occur when comparing the first and subsequent image frames. In addition, in this example our focus is the incremental deformation fields (or the velocity fields) rather than the cumulative displacement fields of an experimentally-captured flow through a bent pipe. Since the tracer particles themselves can be assumed to be rigid in this application, we define the particle rigidity as "hard" (i.e., `MPTPara.parType = 'hard'` ) corresponding to the following lines of code:

```
%%%%% Code mode %%%%%
MPTPara.mode = 'inc'; % {'inc': incremental mode;
                      %  'accum': cumulative mode;
                      %  'dbf': double frame}

%%%%% Particle rigidity %%%%%
MPTPara.parType = 'hard'; % {'hard': hard particle;
                          %  'soft': soft particle}
```

Next, we define how we load the image binary mask file. Here we load a MATLAB mat file as a mask file for all frames. Therefore, we define the parameter `MaskFileLoadingMode = 3;` and provide the path to the store data, which contains a pre-define binary mask.

```
%%%%% Image binary mask file %%%%%
MaskFileLoadingMode = 3; % {0: No mask file
```

```
                            %  1: Load only one mask file for all frames;
                            %  2: Load one mask file for each single frame;
                            %  3: Load a MATLAB mat file for all frames;}

if MaskFileLoadingMode == 3
    im_roi_mask_file_path = '.\img_par2track_pipe\im_roi.mat';
    % TODO: Path of the mat file to be used as the mask file
else
    im_roi_mask_file_path = '';
end
```

### 3.3.2 Particle detection parameters

In this example, we define the particle detection parameters as follows:

```
%%%%% Particle detection parameters %%%%%
%%%%% Bead parameters %%%%%
BeadPara.thres = 0.5;           % Threshold for detecting particles
BeadPara.beadSize = 3;          % Estimated radius of a single particle
BeadPara.minSize = 2;           % Minimum area of a single particle
BeadPara.maxSize = 20;          % Maximum area of a single particle
BeadPara.winSize = [5, 5];      % Default [not used in 2D]
BeadPara.dccd = [1,1];          % Default [not used in 2D]
BeadPara.abc = [1,1];           % Default [not used in 2D]
BeadPara.forloop = 1;           % Default [not used in 2D]
BeadPara.randNoise = 1e-7;      % Default [not used in 2D]
BeadPara.PSF = [];              % PSF function; Example: PSF = fspecial(
                                % 'disk',BeadPara.beadSize-1); % Disk blur
BeadPara.color = 'white';       % Bead color: 'white' -or- 'black'
```

### 3.3.3 Particle linking parameters

Particle linking parameter is defined as:

```
%% SerialTrack particle tracking

%%%%% Multiple particle tracking (MPT) Parameter %%%%%
MPTPara.f_o_s = 30;             % Size of search field: max(|u|,|v|) [px]
MPTPara.n_neighborsMax = 25;    % Max # of neighboring particles
MPTPara.n_neighborsMin = 1;     % Min # of neighboring particles
MPTPara.locSolver = 1;          % Local solver:
                                % 1-topology-based feature;
                                % 2-histogram-based feature first and
                                %   then topology-based feature;
MPTPara.gbSolver = 3;           % Global step solver:
                                % 1-moving least square fitting;
                                % 2-global regularization;
                                % 3-ADMM iterations
MPTPara.smoothness = 1e-2;      % Coefficient of regularization
MPTPara.outlrThres = 2;         % Threshold for removing outliers in TPT
MPTPara.maxIterNum = 20;        % Max ADMM iteration number
```

```
MPTPara.iterStopThres = 1e-2;     % ADMM iteration stopping threshold
MPTPara.strain_n_neighbors = 20; % # of neighboring particles used in
                                  % strain gauge
MPTPara.strain_f_o_s = 60;        % Size of virtual strain gauge [px]
MPTPara.usePrevResults = 0;       % Whether use previous results or not:
                                  % 0-no; 1-yes;
MPTPara.distMissing = 2;          % Distance threshold to check whether
                                  % particle has a match or not [px]
```

### 3.3.4 Merging trajectory segments

We merge tracked trajectory segments using the following strategy. First, every trajectory segment having at least `minTrajSegLength` data points will be extrapolated using the `'pchip'` scheme for a continuous streak line or the `'nearest'` scheme for a Brownian motion. Then we search whether there exist separate segments that can be connected and use the distance threshold ( `distThres` ) measure to connect unintentionally split trajectory segments. Because significant extrapolation errors may occur, we will not merge segments if the frame numbers of their two ends are greater than the user-defined parameter `maxGapTrajSeqLength` .

```
%%%% Postprocessing: merge trajectory segments %%%%%
distThres = 1;            % distance threshold to connect split
                          % trajectory segments [px]
extrapMethod = 'pchip';   % extrapolation scheme to connect split
                          % trajectory segments
                          % suggestion: 'nearest' for Brownian motion
minTrajSegLength = 10;    % the minimum length of trajectory segment that [
   px]
                          % will be extrapolated
maxGapTrajSeqLength = 0; % the max frame # gap between connected
                          % trajectory segments
```

### 3.3.5 Visualization of tracked results

The tracking results of this example (flow through a bent pipe) are summarized in Fig. 16.

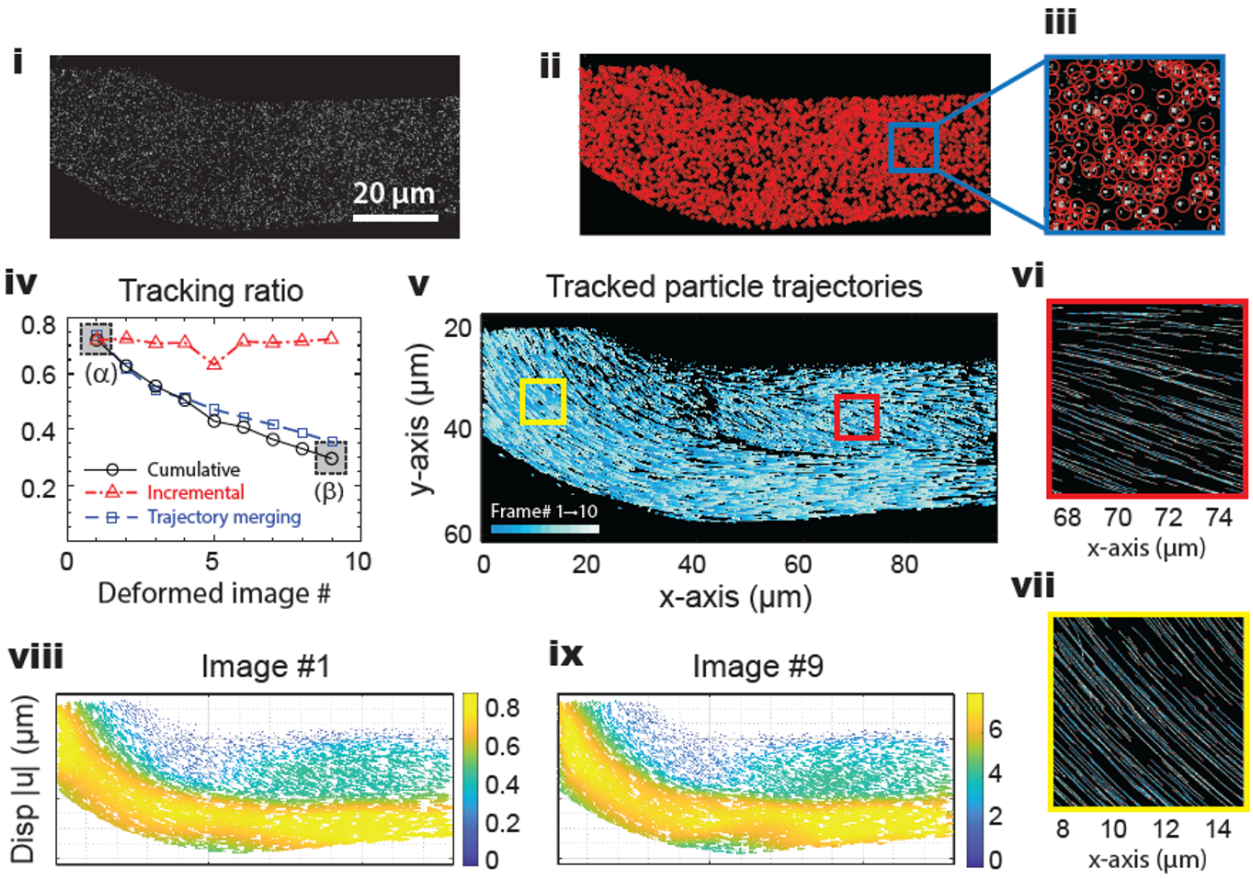


Figure 16: The results of the example of flow through a bent pipe [6]. (i) One typical frame in the time-resolved image sequence. (ii-iii) Detected single particle centroids using the Laplacian of Gaussian filtering technique circled in red. (iv) Particle tracking ratios. (v-vii) Tracked trajectories. (viii-ix) Tracked cumulative displacement fields of the first and ninth frames.

### 3.4 2D double frame tracking mode: flow inside the human lungs

In the "double frame" mode two frames are taken for every time point, with a temporal delay between time points. Each odd number frame is compared to its subsequent even number frame (more details can be found in [9]). Here we aim to reconstruct the air flow vector field inside the primary bronchi of a pair of human lungs as an example. The original data set can be obtained from Janke, et al. [6]. The experimental field of view is highlighted by the green box in Fig. 18(a). As diagrammed in Fig. 17(a), all the image pairs are stored in a folder within MATLAB's working directory. Each image pair is taken with a temporal delay △t and the relative motion between these two frames can be used to calculate the time-resolved velocity fields, as shown in Fig. 17(b).

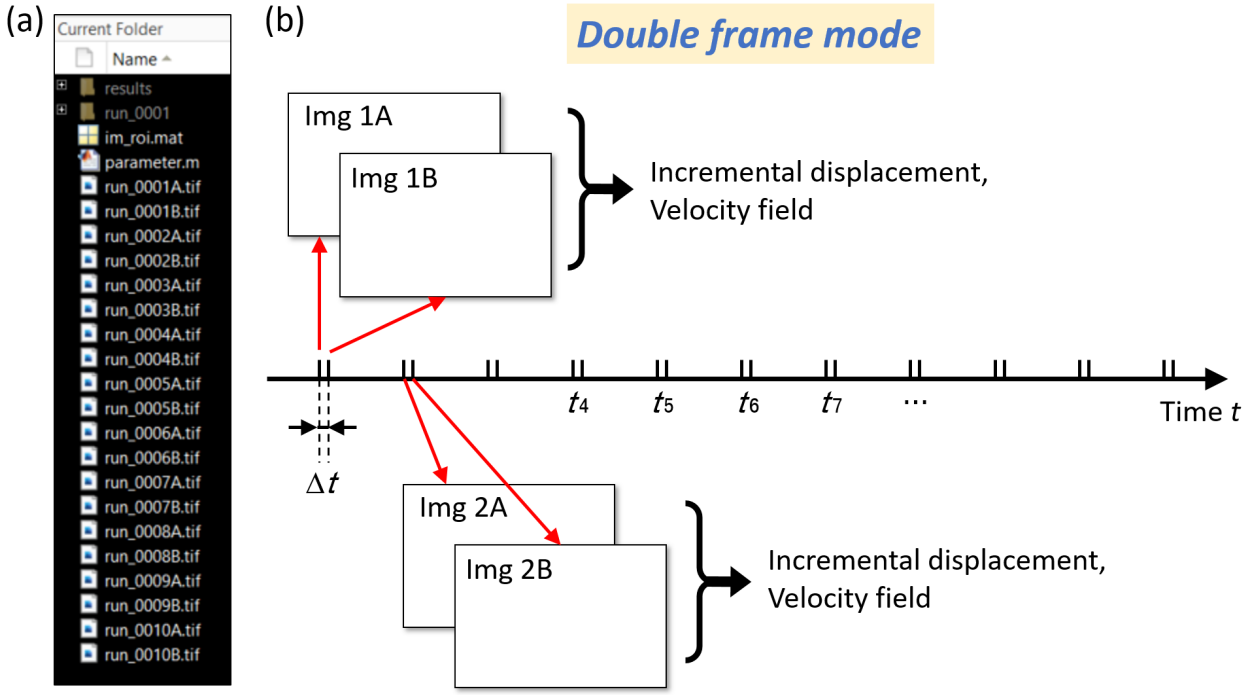


Figure 17: Double frame tracking mode. (a) All image pairs are stored in a folder within MATLAB's working directory. (b) In the double frame mode, two frames are taken within every single exposure with a temporal delay, and each odd number frame is compared to its subsequent even number frame (more details can be found in [9]).

A typical frame of the data set [6] is shown in Fig. 18(b). A small white square box is further enlarged to show that randomly scattered particles can be detected and localized using the following bead detection parameters:

```
%%%%% Particle detection parameters %%%%%
%%%%% Bead Parameter %%%%%
BeadPara.thres = 0.5;           % Threshold for detecting particles
BeadPara.beadSize = 3;          % Estimated radius of a single particle
                                % [px]
BeadPara.minSize = 2;           % Minimum area of a single particle [px^2]
BeadPara.maxSize = 200;         % Maximum area of a single particle [px^2]
BeadPara.winSize = [5, 5];      % Default [not used in 2D]
```

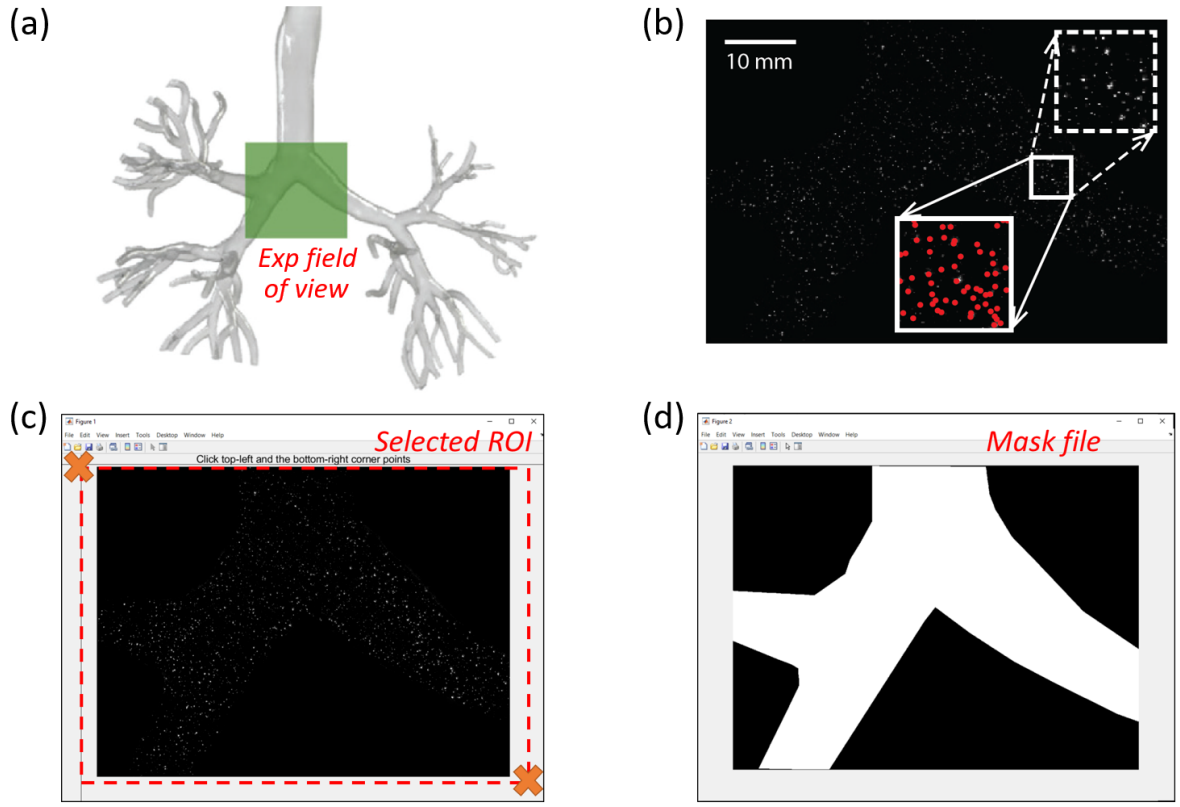


Figure 18: (a) The field of view in the experimental set up to measure the flow field inside the human lungs. (b) One typical captured frame. The small white square is enlarged to show the unprocessed particle images (top) and localized centroids indicated with red dots (bottom), to demonstrate the centroids of randomly scattered particles can be detected. (c) The region of interest (ROI) is defined by manually clicking a top-left and a bottom-right points. (d) An binary image mask can be further applied to refine the motion field reconstruction by accounting for the branched, complex geometry of the human primary bronchi.

```
BeadPara.dccd = [1,1];              % Default [not used in 2D]
BeadPara.abc = [1,1];               % Default [not used in 2D]
BeadPara.forloop = 1;               % Default [not used in 2D]
BeadPara.randNoise = 1e-7;          % Default [not used in 2D]
BeadPara.PSF = [];                  % PSF function;
                                    % Example: PSF = fspecial('disk', ...
                                    % BeadPara.beadSize-1); % Disk blur
BeadPara.color = 'white';           % Bead color: 'white' -or- 'black'
```

After executing the code (“ `Example_main_hardpar_dbf_lung.m` ”), the user is requested to define the region of interest (ROI) by clicking the desired top-left and a right-bottom corner points (see Fig. 18(c)). A binary image mask file (see Fig. 18(d)) can also be provided by specifying its path. **The final tracking area is the intersection of the manually defined ROI and the utilized binary image mask file.**

```
%%%%% Image binary mask file %%%%%
MaskFileLoadingMode = 3; % {0: No mask file
                         %  1: Load only one mask file for all frames;
                         %  2: Load one mask file for each single frame;
                         %  3: Load a MATLAB mat file for all frames;

if MaskFileLoadingMode == 3
    im_roi_mask_file_path = '.\img_par2track_lung\im_roi.mat';
                          % TODO: modify by the user
else
    im_roi_mask_file_path = '';
end
```

Figure 19 summarizes the SerialTrack results of the flow inside human primary bronchi. For each image pair within the double frame mode, the particle tracking ratio is above 50 % (see Fig. 19(a)). An example of the tracked incremental displacement and corresponding velocity fields are shown in (b) and (c), respectively. The averaged velocity and calculated vorticity fields using all ten image pairs are shown in (d) and (e), respectively.

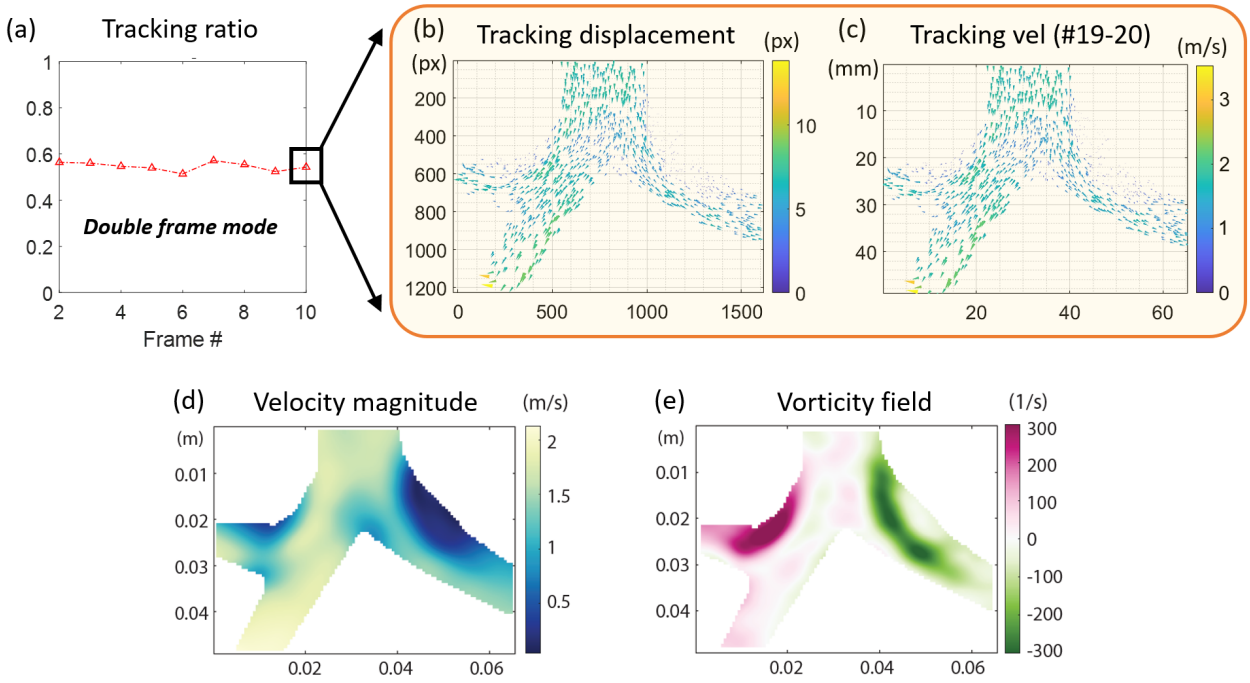


Figure 19: Summary of tracked results of the flow inside the human primary bronchi. (a) The final particle tracking ratios. (b-c) An example of the tracked incremental displacement and corresponding velocity fields. (d-e) The averaged velocity and calculated vorticity fields using all ten image pairs.

### 3.5 2D “Soft particles” - accounting for particle shape distortions in a foam compression test example

Fiducial markers, which SerialTrack identifies as particles, can also be printed or painted on a sample’s surface [10]. Therefore, these particles may have significant shape distortions when the material surface deforms. For example, a circular feature on a 2D sample surface may deform into an ellipse during a uniaxial tension/compression test. This deformation might degrade the particle detection and decrease the tracking accuracy if the shape change is not accounted for. Here, we define particles as “hard” if they are effectively rigid and their shapes do not change throughout the experiment. Alternatively, we call particles “soft” if we assume their shape changes during the deformation. We further assume that the shape distortion of the tracked fiducial marker coincides with local deformation gradients, as shown in Fig. 20(b).

We therefore implement an optional algorithm which accounts for the effect of fiducial marker shape distortions. The “soft” tracking algorithm uses reconstructed deformation field from the immediately previous frame to warp the current configuration images, then detects and localizes particles in the synthetically warped images. This is implemented within Algorithm 2 iterations (see our original paper Algorithm 2).

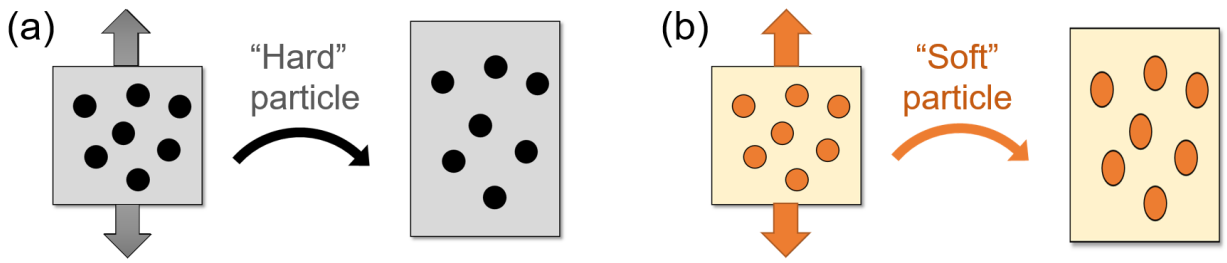


Figure 20: (a) We consider the particles to be “hard” if the shape is effectively rigid and invariant throughout the experiment. (b) Alternatively, we consider the particles to be “soft” if the particle shape changes, in which case we assume that their shape change are governed by the deformation gradient of the surrounding field.

The user can open and execute the mfile script “ `Example_main_softpar_accum_foam.m` ” which is stored in the subfolder “Example_main_files”. Similarly to Section 3.1, the mfile header describes the basic information for the code.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% SerialTrack execute main file
% ===================================================
% Dimension:              2D
% Particle rigidity:      soft
% Tracking mode:          cumulative
% Syn or Exp:             exp
% Deformation mode:       foam uniaxial compression
%
% ===================================================
% Author: Jin Yang, Ph.D.
```

```
% Email: jyang526@wisc.edu -or-  aldicdvc@gmail.com
% Date: 02/2022
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

### 3.5.1 User-defined parameters

In this section, we define the problem variables for the soft foam test case.

```
%% user-defined parameters %%%%%

%%%%% Problem dimension and units %%%%%
MPTPara.DIM = 2;    % problem dimension
MPTPara.xstep = 1;  % unit: um/px
MPTPara.tstep = 1;  % unit: us -or- 1/frame

%%%%% Code mode %%%%%
MPTPara.mode = 'accum'; % {'inc': incremental mode;
                        %  'accum': cumulative mode;
                        %  'dbf': double frame}

MPTPara.parType = 'soft'; % {'hard': hard particle;
                          %  'soft': soft particle}

disp('-----------------------------------------------');
disp(['Dimention: ',num2str(MPTPara.DIM)]);
disp(['Tracking mode: ',MPTPara.mode]);
disp(['Particle type: ',MPTPara.parType]);
disp('-----------------------------------------------'); fprintf('\n');

%%%%% Image binary mask file %%%%%
MaskFileLoadingMode = 0; % {0: No mask file
                         %  1: Load only one mask file for all frames;
                         %  2: Load one mask file for each single frame;
                         %  3: Load a MATLAB mat file for all frames;
```

### 3.5.2 Particle detection and localization parameters

In this case (black particles) the code uses the MATLAB "regionprops" function to extract particle (or blob) centroids. The particle size can be various ( `BeadPara.beadSize = 0;` ) but need to be within a minimum and a maximum radii based on how blob selection is handled. The particle detection parameters are defined as

```
%%%%% Particle detection and localization parameters %%%%%
%%%%%%%%%% Elastomeric open-cell foam %%%%%%%%%%%%
%%%%% Bead Parameter %%%%%
BeadPara.thres = 0.3;           % Threshold for detecting particles
BeadPara.beadSize = 0;          % Estimated area of a single particle [px]
BeadPara.minSize = 3;           % Minimum area of a single particle [px^2]
BeadPara.maxSize = 200;         % Maximum area of a single particle [px^2]
BeadPara.winSize = [5, 5];      % Default [not used in 2D]
```

```
BeadPara.dccd = [1,1];          % Default [not used in 2D]
BeadPara.abc = [1,1];           % Default [not used in 2D]
BeadPara.forloop = 1;           % Default [not used in 2D]
BeadPara.randNoise = 1e-7;      % Default [not used in 2D]
BeadPara.PSF = [];              % PSF function;
                                % Example: PSF = fspecial('disk', ...
                                % BeadPara.beadSize-1 ); % Disk blur
BeadPara.color = 'black';       % Bead color: 'white' -or- 'black'
```

### 3.5.3 Particle linking parameters

The particle linking parameters are summarized below:

```
%% SerialTrack particle tracking

%%%%% Multiple particle tracking (MPT) Parameter %%%%%
MPTPara.f_o_s = 30;               % Size of search field: max(|u|,|v|) [px]
MPTPara.n_neighborsMax = 25;      % Max # of neighboring particles
MPTPara.n_neighborsMin = 1;       % Min # of neighboring particles
MPTPara.locSolver = 1;            % Local solver: 1-topology-based feature;
                                  % 2-histogram-based feature first
                                  % and then topology-based feature;
MPTPara.gbSolver = 3;             % Global step solver:
                                  % 1-moving least square fitting;
                                  % 2-global regularization;
                                  % 3-ADMM iterations
MPTPara.smoothness = 1e-1;        % Coefficient of regularization
MPTPara.outlrThres = 2;           % Threshold for removing outliers in TPT
MPTPara.maxIterNum = 20;          % Max ADMM iteration number
MPTPara.iterStopThres = 1e-3;     % ADMM iteration stopping threshold
MPTPara.strain_n_neighbors = 20; % # of neighboring particles used in
                                  % strain gauge
MPTPara.strain_f_o_s = 60;        % Size of virtual strain gauge [px]
MPTPara.usePrevResults = 1;       % Use previous results? 0-no; 1-yes;
MPTPara.distMissing = 10;         % Distance threshold to check whether
                                  % particle has a match or not [px]


%%%% Postprocessing: merge trajectory segments %%%%%
distThres = 1; % distance threshold to connect trajectory segments [px]
extrapMethod = 'pchip';  % extrapolation scheme to connect split
                         % trajectory segments
                         % suggestion: 'nearest' for Brownian motion
minTrajSegLength = 20;   % the minimum length of trajectory segment that
                         % will be extrapolated [px]
```

### 3.5.4 Executing the SerialTrack particle tracking code for soft particles

We only implemented the cumulative tracking mode for the soft particles case. Significant particle shape distortions between frames only appear in the presence of significant material strains, which

are atypical in incremental and double frame modes. The lines of code to set up the SerialTrack particle tracking for soft particle mode are as follows:

```
%%%%% Execute SerialTrack particle tracking %%%%%
if strcmp(MPTPara.mode,'inc')==1
    disp('Not implemented yet.');
    % run_Serial_MPT_2D_softpar_inc;
elseif strcmp(MPTPara.mode,'accum')==1
    run_Serial_MPT_2D_softpar_accum;
elseif strcmp(MPTPara.mode,'dbf')==1
    disp('Not implemented yet.');
    % run_Serial_MPT_2D_hardpar_dbf;
end
```

After executing the mfile, the user is prompted to select the image folder from which to load the images (see Fig. 21). This step is same as before. Here we show a uniaxial compression test of a hyperelastic foam material [10]. The data set "img_foam_cp50_0_001_selected" can be downloaded via the MINDS@UW open access institutional data repository [4].

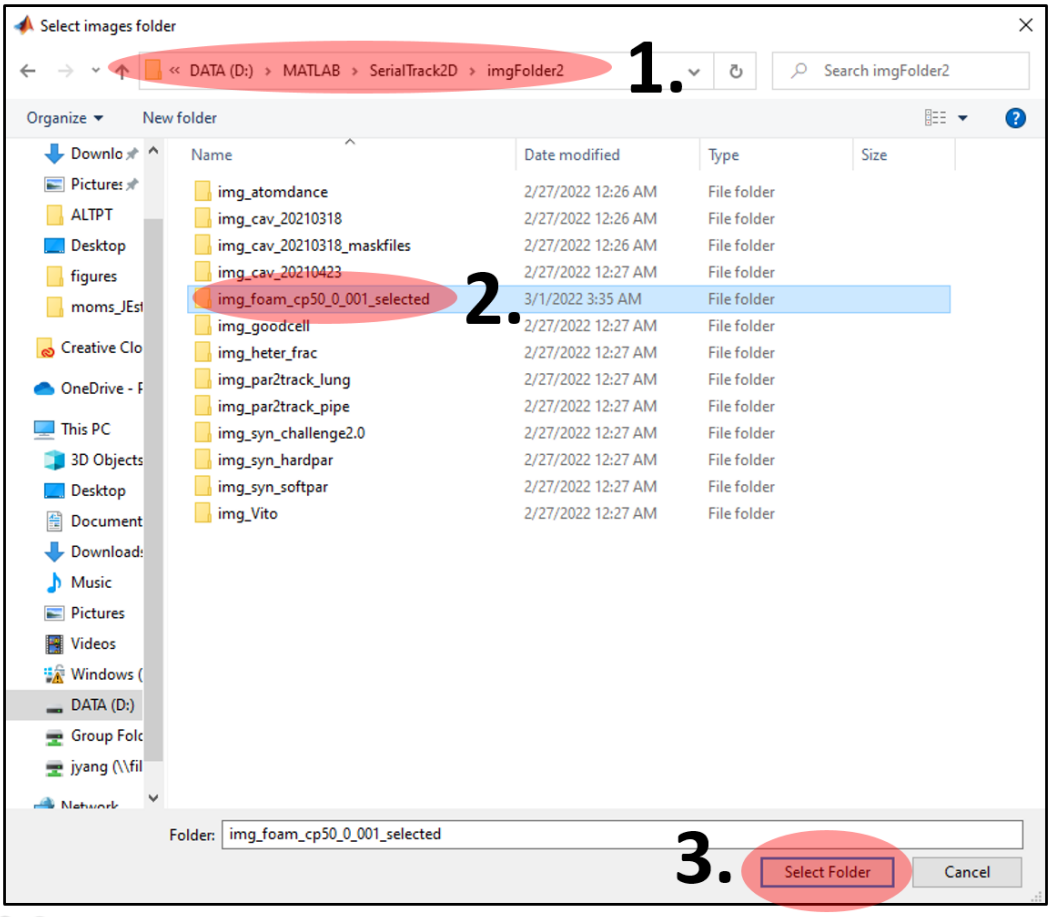


Figure 21: Images can be loaded by manually selecting an image folder where all the frames are stored.

A window will pop up for the user to define the region of interest (ROI). A ROI can be defined by manually clicking the desired top-left and a bottom-right corner points as shown in Fig. 22. Then, the image sequence will be post-processed to track the particle motions, and the MATLAB command window will display the kind of information as shown in Fig. 23.

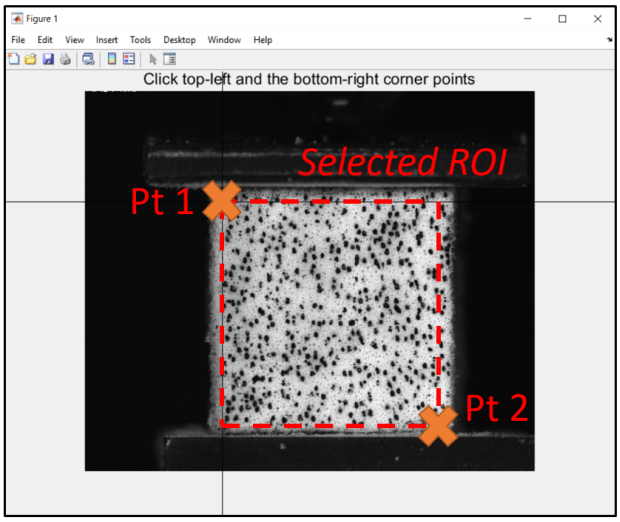


Figure 22: An ROI can be defined by manually clicking the top-left and bottom-right corner points within the image.

### 3.6 Tracking detected 2D particle coordinates

We also provide an example execution file for users to track a set of coordinates of 2D detected particles in each frame. The centroids of these particles can be obtained from other software or code packages, which is not necessary to use our included particle detection algorithms. Or they can be any feature points instead of particles. The goal of this section is to demonstrate how the user can link and track these particle coordinates using the SerialTrack algorithm.

We provide two example mfiles called “Example_main_2D_hardpar_accum_coords_only.m” and “Example_main_2D_hardpar_inc_coords_only.m” and they are stored in subfolder “./SerialTrack2D/Example_main_files/”. Here, “accum” and “inc” mean “cumulative” and “incremental” tracking modes, respectively.

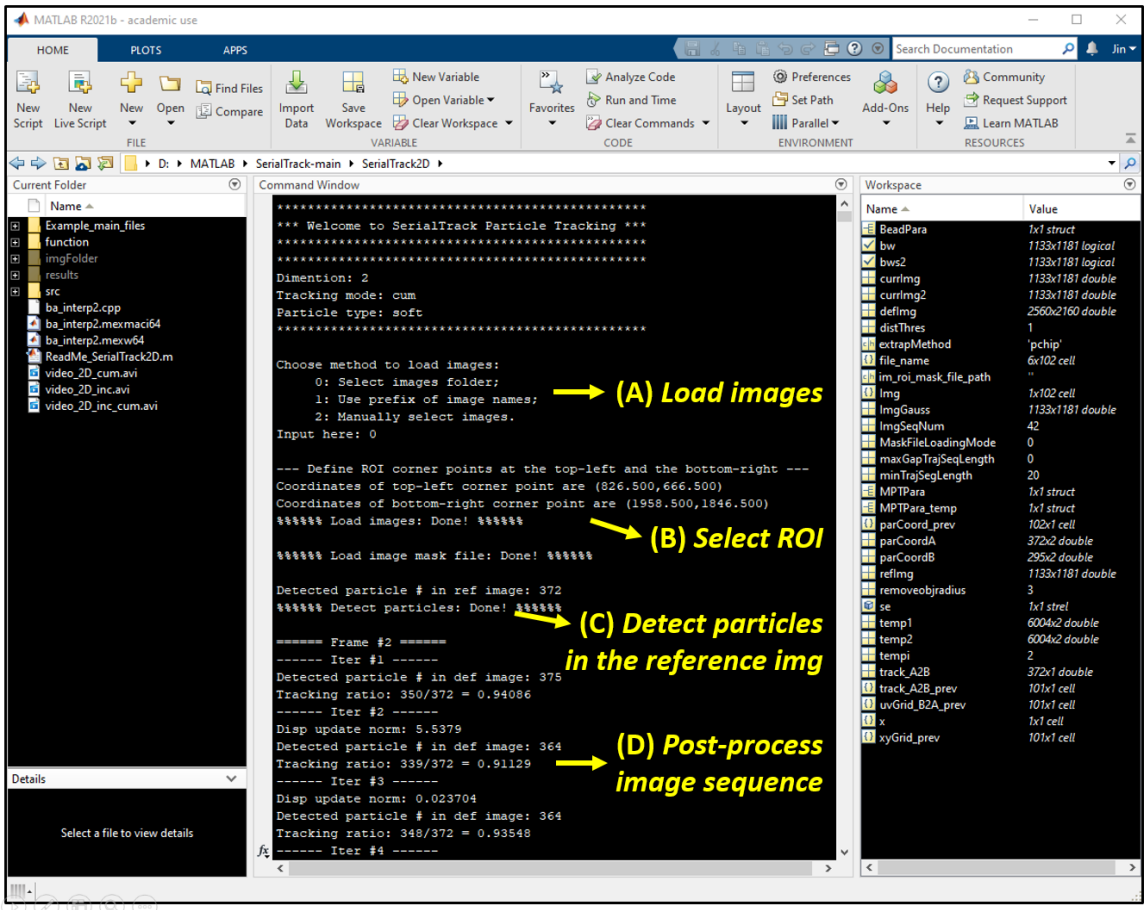


Figure 23: The screen shot of the MATLAB integrated development environment window during code execution.

# 4 SerialTrack 3D Examples

Similarly to the 2D cases, 3D particle centroids can be tracked and their 3D (volumetric) deformation fields can be found by comparing deformed volumetric image stacks to the reference volumetric image stacks. Several examples of MATLAB execution main files for 3D examples are stored in the subfolder "./Example_main_files" (see Fig. 24(A)). The required changes to the 2D version of SerialTrack to handle 3D deformations are twofold. First, image stacks need to be transformed into volumetric matrices and stored as MATLAB .mat files (see Fig. 24(B)), which can be pre-processed using the script "GenerateVolmatfile.m" (see Fig. 24(C)). Second, we need to modify the individual particle descriptors for 3D topology by adding a second angular vector in the feature vector (see main manuscript Figure 2(c-d)).

## 4.1 Transforming image stacks to MATLAB .mat files

To transform an image stack into a volumetric image, the user should store the image stack as a series of individual 2D images in a subfolder within the SerialTrack folder as shown in Fig. 25. Next,

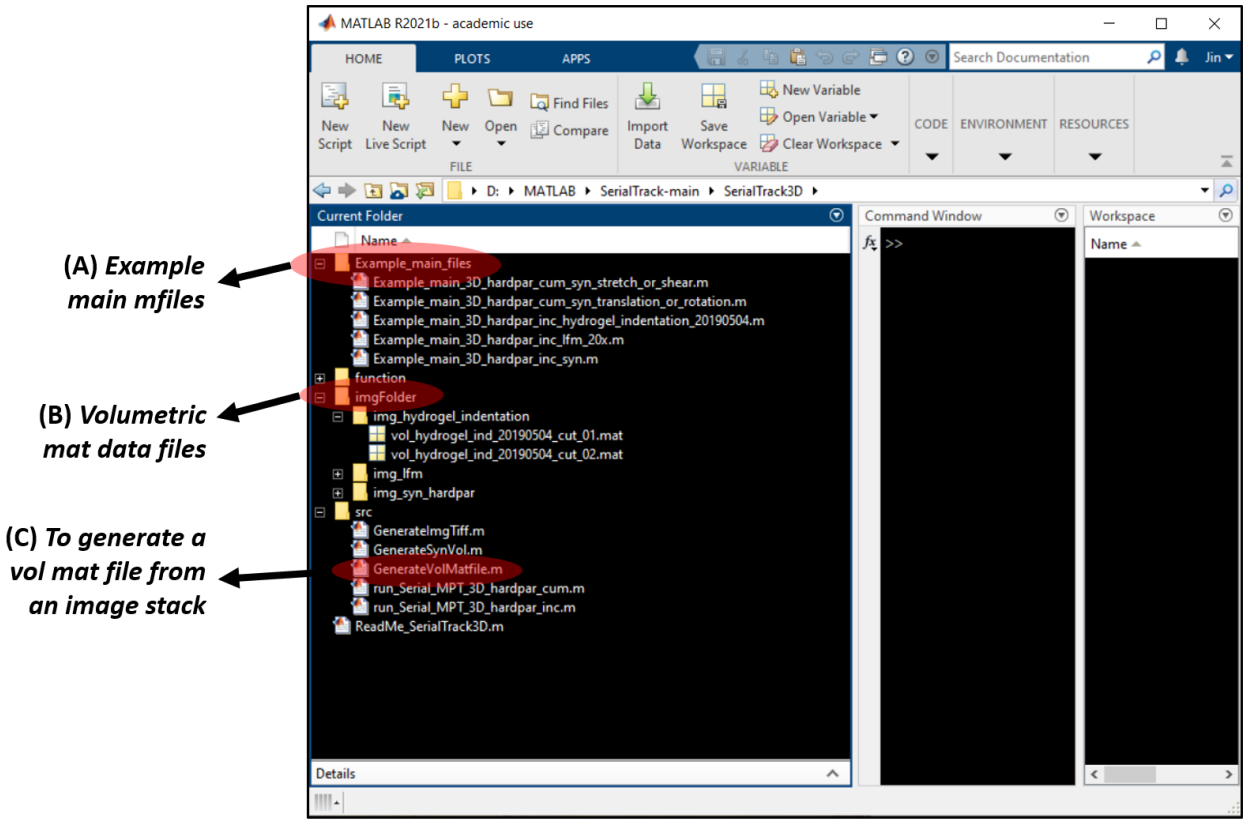


Figure 24: The directory structure for an example 3D (volumetric) case.

the user needs to modify several `TODO` lines in the "GenerateVolMatfile.m" script to transform an image stack into a MATLAB matrix in memory, and save them out to disk as .mat files. All these "TODO" steps will be explained below.

**First**, the user needs to add all of the stored images for a image sequence to the MATLAB working directory and in the same folder as shown in Fig. 25. Each image in the stack should be named sequentially, with leading zeros as needed for numbering.

```
% TODO: Use your image prefix and extension
files = dir('./vol_stretch_1001_tiff/vol_*.tif');
```

We use the " `imread` " function to read each image stack. If they are grayscale images, directly applying the " `imread` " function will work well. If they are RGB images, the user needs to apply the " `rgb2gray` " function after using the " `imread` " function.

```
% TODO: check whether your images are rgb or grayscale images
% If it's a RGB image, please use "rgb2gray()" function:
% f = rgb2gray(imread(im{tempi},1));
f = imread(im{tempi},1);
```

The user also needs to retrieve the bit depth of the image stack and modify the following line correspondingly.

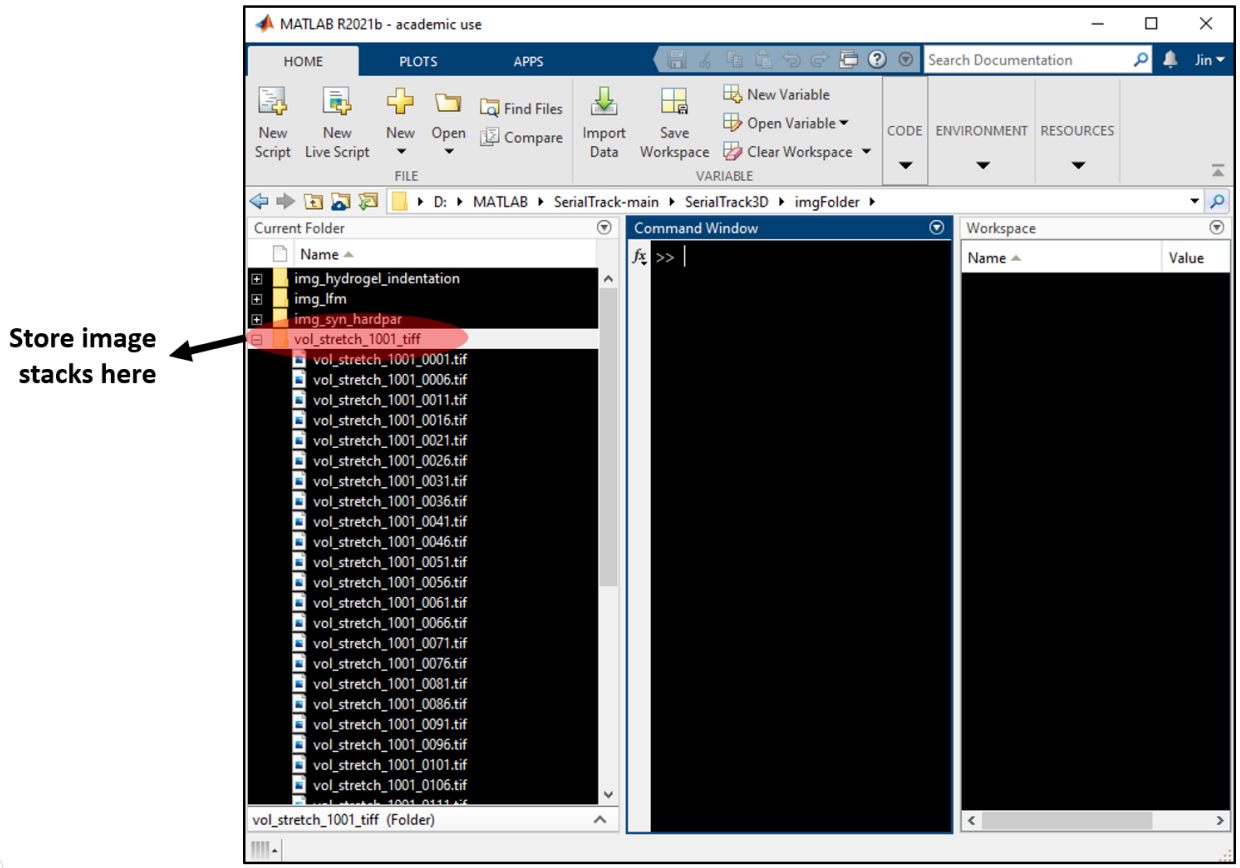


Figure 25: Sequentially numbered images for volume stacks are stored in a separate folder for each experiment.

```
% TODO: change "uint8" to "uint'x'" if your images are not 8-bit images.
vol{1} = uint8(permute(voltemp,[2,1,3]));
```

The user needs to provide the file name to save the generated volumetric data set.

```
% TODO: provide a matlab file name to save the generated "vol" matrix
save 'vol_stretch_1001.mat' vol ;
```

Finally, the user can plot the generated volumetric image stack using the " `imagesc3` " function (included in the SerialTrack code package), as shown in Fig. 26.

```
% Show generated volumetric image stack
figure, imagesc3(vol{1});
% or: imagesc3D(permute(double(vol{1}),[2,1,3]));
```

## 4.2 Synthetic 3D volumetric images and applying known deformations

We include the MATLAB code to synthesize 3D volumetric bead pattern images and apply imposed deformation fields. The user can open the mfile stored in "./SerialTrack3D/src/" and run

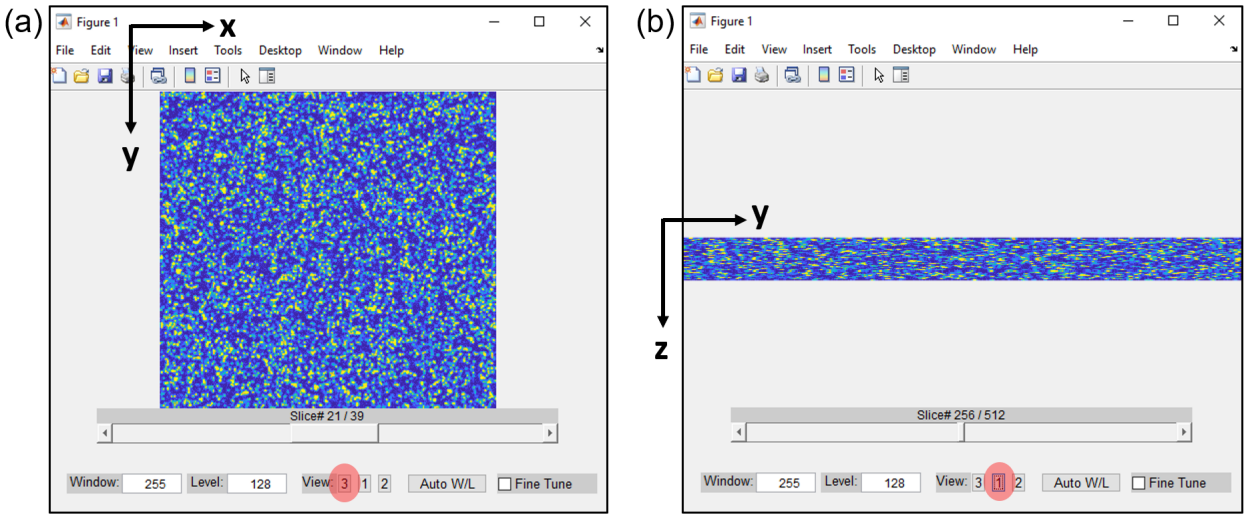


Figure 26: Generated volumetric matrix can be visualized using the " `imagesc3` " function.

"GenerateSynVol.m". This file can simulate four different types of uniform deformations ( `DefType` ): **(a) rigid body translations**, **(b) rigid body rotations around z-axis**, **(c) uniaxial stretches**, and **(d) simple shear**, as shown in the main manuscript Fig. 3(e-h). The user can simulate different particle seeding densities by changing the variable `seedingDensity` . In addition, the desired output volumetric image size can be customized by adjusting the value of the variable `sizeI` .

All the synthetic datasets can be downloaded from [4], and they can be further tested using the mfile "Example_main_3D_hardpar_inc_sym.m" stored in "./SerialTrack3D/Example_main_files/". The following lines need to be modify correspondingly.

```
%%%%% SerialTrack path %%%%%
SerialTrackPath = pwd; % % TODO: modify the path of "SerialTrack3D";
% Example: SerialTrackPath = 'D:\MATLAB\SerialTrack-main\SerialTrack3D';

%%%%% Synthetic cases %%%%%
DefType = 'stretch';          % Loading type:  {'translation','stretch',
                              %'simpleshear','rotation'}
SeedingDensityType = 2;       % Particle seeding density type {1,2,3,4} =
                              % {10,100,300,1000}*1e-6 particles per voxel
fileNameAll = 'vol_*.mat';    % file name(s)
```

### 4.3 3D dense particle tracking: hydrogel indentation

We provide a real 3D volumetric experimental example to demonstrate the SerialTrack-3D code. The original data set can be downloaded from [4]. This is a 3D hydrogel indentation experiment where a steel sphere indents a polyacrylamide hydrogel from the top surface under its own weight. For more information about the experiment see [11].

The user can execute the mfile "Example_main_3D_hardpar_inc_hydrogel_indentation.m" to test this example, which is stored in "./SerialTrack3D/Example_main_files".

### 4.3.1 Initialization of the 3D hydrogel indentation example

The code block below outline the initialization settings used for the 3D hydrogel deformation example.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% SerialTrack execute main file
% ================================================
% Dimension:              3D
% Particle rigidity:      hard
% Tracking mode:          incremental
% Syn or Exp:             exp
% Deformation mode:       hydrogel indentation
%
% ================================================
% Author: Jin Yang, Ph.D.
% Email: jyang526@wisc.edu -or-  aldicdvc@gmail.com
% Date: 02/2022
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Initialization
close all; clear all; clc; clearvars -global
disp('************************************************');
disp('*** Welcome to SerialTrack Particle Tracking ***');
disp('************************************************');
addpath( './function/','./src/','./Scatter2Grid3D' );


%% user-defined parameters %%%%%

%%%%% Problem dimension and units %%%%%
MPTPara.DIM = 3;      % problem dimension
MPTPara.xstep = 0.4;  % unit: um/px
MPTPara.ystep = 0.4;  % unit: um/px
MPTPara.zstep = 0.4;  % unit: um/px
MPTPara.tstep = 1;    % unit: us -or- 1/frame

%%%%% Code mode %%%%%
MPTPara.mode = 'inc'; % {'inc': incremental mode;
                      %  'accum': cumulative mode}

%%%%% Particle rigidity %%%%%
MPTPara.parType = 'hard'; % {'hard': hard particle;
                          %  'soft': soft particle}

disp('************************************************');
disp(['Dimention: ',num2str(MPTPara.DIM)]);
disp(['Tracking mode: ',MPTPara.mode]);
disp(['Particle type: ',MPTPara.parType]);
disp('************************************************'); fprintf('\n');

%%%%% SerialTrack path %%%%%
```

```
SerialTrackPath = pwd; % % TODO: modify the path of "SerialTrack3D";
% Example: SerialTrackPath = 'D:\MATLAB\SerialTrack-main\SerialTrack3D';

%%%%% Volumetric image path %%%%%
fileNameAll = 'vol_hydrogel_ind_20190504_cut_0*.mat';
fileFolder = './imgFolder/img_hydrogel_indentation/';
```

### 4.3.2 Particle detection and localization parameters

For the volumetric particle detection and localization, there are two methods implemented in SerialTrack. Method "1" corresponds the the method used in the TPT algorithm (see [7]), which uses a threshold-based segmentation to get particle blobs, and a radial projection center-finding to localize the centroid. Method "2" uses a Laplacian of gradient-base filter to segment blobs, followed by a least squares fit of a 2nd order polynomial to the intensity peak to localize the centroid [5, 6]. In general "1" is faster but less robust. For a new dataset it is best to test both (or implement a new method entirely) and iteratively adjust settings, since particle detection and localization is a critical step, but can be relatively problem-specific.

```

%%%%% Image binary mask file %%%%%
im_roi_mask_file_path = ''; % TODO: leave it as none if
                            % there is no mask file

%%%%% particle detection and localization method %%%%%
BeadPara.detectionMethod = 2; % Particle detection method:
                    % 1: TPT (blob finding + radial projection);
                    % 2: TracTrac (LoG blob finding + lsq fit of gaussian)

%%%%% Bead Parameter %%%%%
BeadPara.thres = 0.1;          % Threshold for detecting particles
BeadPara.beadSize = 3;         % Estimated radius of a single particle
BeadPara.minSize = 3;          % Minimum volume of a single particle [px^3]
BeadPara.maxSize = 20;         % Maximum volume of a single particle [px^3]
BeadPara.winSize = [5,5,5];    % Default [not used for method 2]
BeadPara.dccd = [1,1,1];       % Default [not used for method 2]
BeadPara.abc = [1,1,1];        % Default [not used for method 2]
BeadPara.forloop = 1;          % Default [not used for method 2]
BeadPara.randNoise = 1e-7;     % Default small number for background noise
BeadPara.PSF = [];             % PSF function; Example:
               % PSF = fspecial('disk', BeadPara.beadSize-1 ); % Disk blur
BeadPara.color = 'white';      % Foreground (particle) color:
                               % options, 'white' or 'black'
```

### 4.3.3 Particle linking parameters

As previously mentioned, the algorithm constructs a 3D feature descriptor for each individual particle. Analogous to the 2D descriptor, 3D radial distances (r), polar angles ($\theta$), and azimuthal angles ($\phi$) of the stored k neighboring particles are used to construct particle descriptors for each particle.

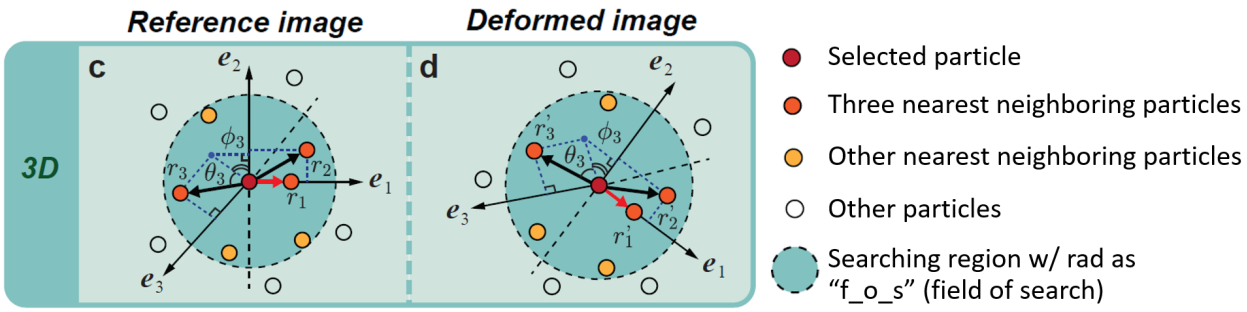


Figure 27: A diagram outlining the 3D descriptor generation process. (a) The $k$ radii and angles to nearest neighbor particles within the search distance ("f_o_s") for each particle. (b) The same computation is performed on the deformed image. By simultaneously minimizing the Euclidean distance for angular and distance mismatch we achieve a fast linking procedure that is scale and rotation invariant, and thus robust under most kinematically admissible deformations.

To establish a coordinate system, $\{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\}$, for each particle, we define the first nearest neighbor particle direction as the $\mathbf{e}_1$ axis. The $\mathbf{e}_3$ direction is defined to be perpendicular to the first and second nearest neighbor particles, and must satisfy $\mathbf{e}_3 \cdot \mathbf{r}_3 > 0$ where $\mathbf{r}_3$ is the third nearest neighbor particle direction. The $\mathbf{e}_2$ axis is defined as $\mathbf{e}_3 \times \mathbf{e}_1$ where " $\times$ " is the cross product. The radial distance (r) feature is the Euclidean interparticle distance, which is normalized by the first nearest neighbor particle distance in the descriptor.

```
%%%%% Multiple particle tracking (MPT) Parameter %%%%%
MPTPara.f_o_s = 60;                % Size of search: max(|u|,|v|,|w|) [px]
MPTPara.n_neighborsMax = 25;       % Max # of neighboring particles
MPTPara.n_neighborsMin = 1;        % Min # of neighboring particles
MPTPara.gbSolver = 2;              % Global step solver:
                                   % 1-moving least square fitting;
                                   % 2-global regularization;
                                   % 3-ADMM iterations
MPTPara.smoothness = 1e-1;         % Coefficient of regularization
MPTPara.outlrThres = 5;            % Threshold for removing outliers in MPT
MPTPara.maxIterNum = 20;           % Max ADMM iteration number
MPTPara.iterStopThres = 1e-3;      % ADMM iteration stopping threshold
MPTPara.strain_n_neighbors = 20; % # of neighboring particles used in
                                   % strain gauge
MPTPara.strain_f_o_s = 60;         % Size of virtual strain gauge [px]
MPTPara.usePrevResults = 0;        % Whether use previous results or not:
                                   % 0-no; 1-yes;
MPTPara.distMissing = 5;           % Distance threshold to check whether
                                   % particle has a match or not [px]
```

### 4.3.4 Merging trajectory segments

```
%%%% Postprocessing: merge trajectory segments %%%%%
distThres = 1;                % distance threshold to connect split
                              % trajectory segments [px]
extrapMethod = 'pchip';       % extrapolation scheme to connect split
```

```
                                % trajectory segments
                                % suggestion: 'nearest' for Brownian motion
minTrajSegLength = 10;          % the minimum length of trajectory segment
                                % that will be extrapolate [px]
maxGapTrajSeqLength = 0;        % the max frame# gap between connected
                                % trajectory segments
```

### 4.3.5 Executing the SerialTrack particle tracking code

```
%%%%% Execute SerialTrack particle tracking %%%%%
if strcmp(MPTPara.mode,'inc')==1
    run_Serial_MPT_3D_hardpar_inc;
elseif strcmp(MPTPara.mode,'accum')==1
    run_Serial_MPT_3D_hardpar_accum;
end
```

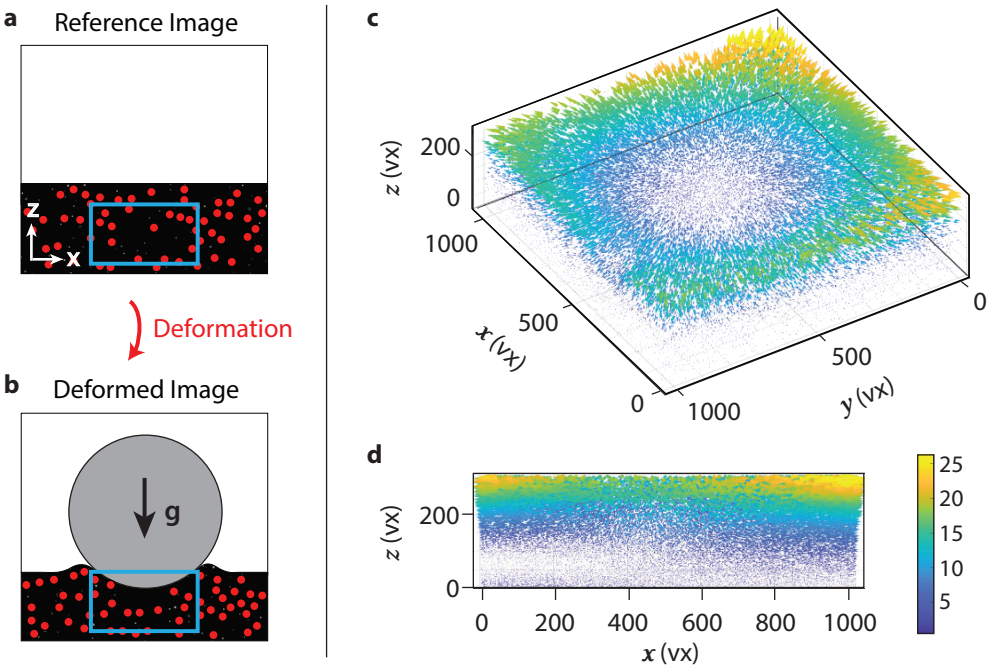


Figure 28: A 3D spherical hydrogel indentation example. (a-b) Sketch of the reference and deformed configurations of the spherical indentation experiment. (c-d) 3D cone plot and xz-plane projection of the tracked 3D deformation.

## 4.4 Tracking detected 3D particle coordinates

We also provide an example execution file for users to track a set of coordinates of 3D detected particles in each frame. The centroids of these particles can be obtained from other software or code packages, which is not necessary to use our included particle detection algorithms. Or they can be any feature points instead of particles. The goal of this section is to demonstrate how the user can link and track these particle coordinates using the SerialTrack algorithm.

We provide two example mfiles called “Example_main_3D_hardpar_accum_coords_only.m” and “Example_main_3D_hardpar_inc_coords_only.m” and they are stored in subfolder “./SerialTrack3D/Example_main_files/”. Here, “accum” and “inc” mean “cumulative” and “incremental” tracking modes, respectively.

## Acknowledgments

The authors thank the U.S. Office of Naval Research for research support under the “PANTHER” program, award number N000142112044 through Dr. Timothy Bentley. This research was performed in part while A.K.L. held an NRC Research Associateship award with Aaron Forster at the National Institute of Standards and Technology. We also would like the thank the reviewer from SoftwareX for providing valuable feedback in the construction of this manual.

## References


[1] SerialTrack paper manuscript ResearchGate link. https://www.researchgate.net/publication/359434795_SerialTrack_ScalE_and_Rotation_Invariant_Augmented_Lagrangian_Particle_Tracking. Accessed: 2022-06-01.

[2] SerialTrack paper manuscript arxiv link. https://arxiv.org/abs/2203.12573. Accessed: 2022-06-01.

[3] SerialTrack code github repository. https://github.com/FranckLab/SerialTrack. Accessed: 2022-06-01.

[4] MINDS@UW open access institutional data repository. https://minds.wisconsin.edu/handle/1793/82901. Accessed: 2022-07-05.

[5] J. Heyman. TracTrac: A fast multi-object tracking algorithm for motion estimation. *Computers & Geosciences*, 128:11–18, 2019.

[6] T. Janke, R. Schwarze, and K. Bauer. Part2Track: A MATLAB package for double frame and time resolved Particle Tracking Velocimetry. *SoftwareX*, 11:100413, 2020.

[7] M. Patel, S. E Leggett, A. K Landauer, I. Y Wong, and C. Franck. Rapid, topology-based particle tracking for high-resolution measurements of large complex 3D motion fields. *Scientific Reports*, 8(1):1–14, 2018.

[8] L. Hazlett, A. K Landauer, M. Patel, H. A Witt, J. Yang, J. S Reichner, and C. Franck. Epifluorescence-based three-dimensional traction force microscopy. *Scientific Reports*, 10:16599, 2020.

[9] M. Raffel, C. E. Willert, S. T. Wereley, and J. Kompenhans. *PIV Recording Techniques*, pages 97–121. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[10] J. Yang, J.-L. Tao, and C. Franck. Smart Digital Image Correlation Patterns via 3D Printing. *Experimental Mechanics*, 61:1181–1191, 2021.

[11] J Yang, L Hazlett, A. K. Landauer, and C Franck. Augmented Lagrangian Digital Volume Correlation (ALDVC). *Experimental Mechanics*, 60(9):1205–1223, 2020.