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

Jin Yang[1], Yue Yin[1,2], 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 5, 2022

Contents

1 Introduction

This code manual is to help guide the users to test our new particle tracking algorithm – ***ScalE and Rotation Invariant Augmented Lagrangian Particle Tracking (SerialTrack)*** – to accurately resolve both ***2D & 3D*** large deformation and rotational motion fields. SerialTrack method takes advantage of both local and global particle tracking algorithms. It builds an iterative scale and rotation invariant topology-based feature for each particle within a multi-scale tracking algorithm. The global kinematic compatibility condition is applied as a global augmented Lagrangian constraint to enhance the tracking accuracy.

To help users to test the SerialTrack method and apply this method to their research projects, an open-source MATLAB code package is provided. The SerialTrack code follows the workflow as shown in Fig. 1. The particle tracking parameter setup is broadly categorized by problem's ***dimension***, ***particle rigidity***, and ***tracking mode***. For all tracking parameters, tracking process includes three main procedures: (i) ***particle detection***, (ii) ***particle linking***, and (iii) ***post-processing***. For more details of the SerialTrack method and a summary of other state-of-art particle tracking methods, please check our manuscript (full text can also be requested via the ResearchGate link [1] or the arxiv link [2]).

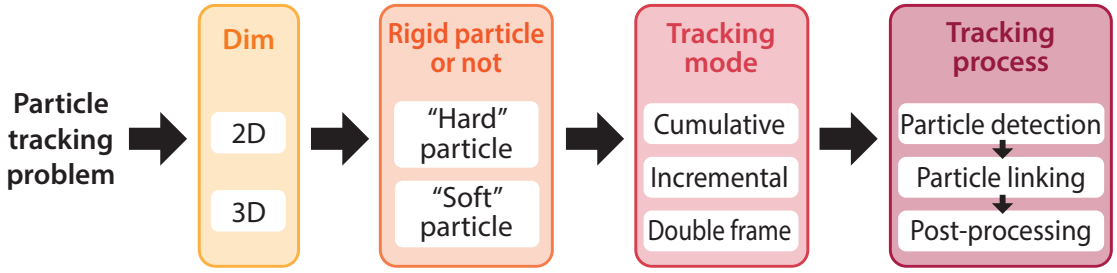

Figure 1: Workflow of SerialTrack. The particle tracking setup is most broadly categorized by problem dimensionality as 2D (pixel image) or 3D (volumetric voxel image). Hard vs. soft particles further specify where particle shape warping is included during the solution. The tracking mode defines the scheme for selecting images from the experimental image sequence. Finally, the particles detection, linking, and post-processing strategies are specified given the experimental configuration and desired output data.

Particle tracking is not the only method to track motions or deformations. Besides tracking individual single particles, there are also subset-based tracking methods including Particle Image Velocimetry (PIV), 2D Digital Image Correlation (DIC), 3D stereo Digital Image Correlation (3D stereo-DIC), and 3D Digital Volume Correlation (DVC) methods. More details about these alternative methods can be found in our main manuscript Table 2.

2 Code Installation

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

To install the code, please download and unzip the code folder, and add the folder to MATLAB working path by right clicking the folder and select "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).

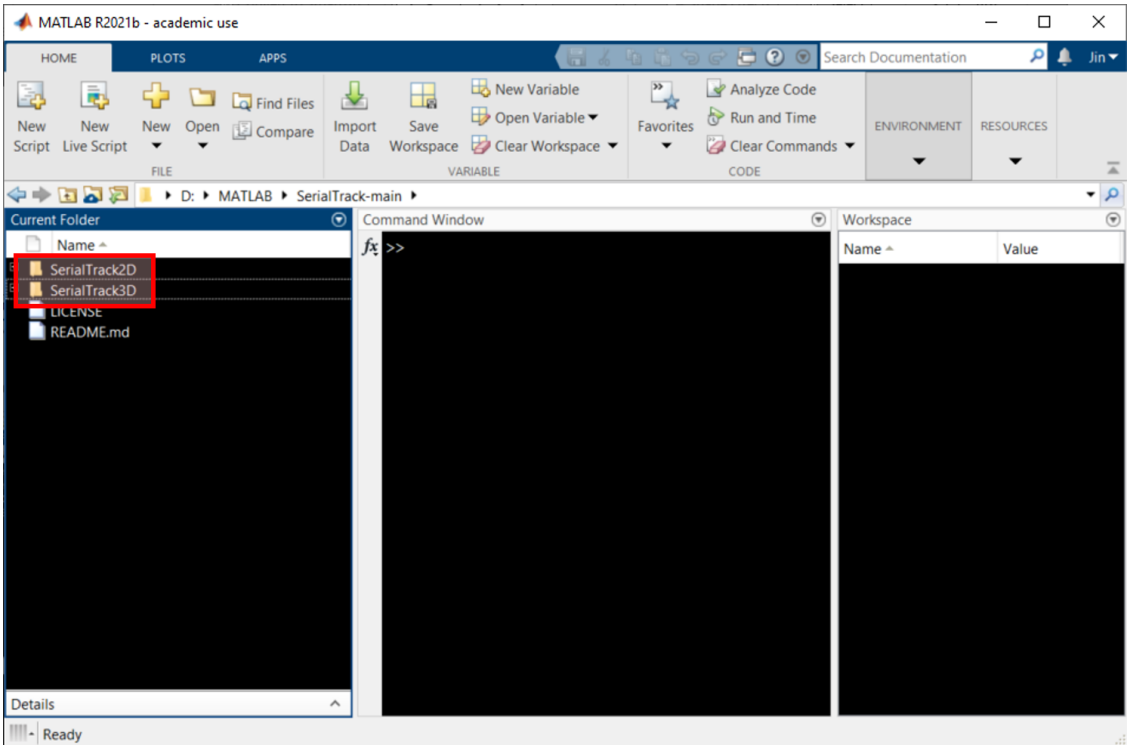

Figure 2: The main path of the SerialTrack MATLAB code package. Serial code package includes two subfolders, "SerialTrack2D" and "SerialTrack3D", to track 2D and 3D motions/deformations, respectively.

For users to test this new SerialTrack particle tracking altorithm, we provide several examples of main files in two subfolders ("SerialTrack2D" and "SerialTrack3D") which can be executed directly (see Fig. 3 yellow box). We also provide example data sets which can be downloaded on the MINDS@UW open access institutional data repository [4]. These data sets or other user provided experimental data sets (at least two frames) can be stored in another subfolder, i.e., "imgFolder" as shown in Fig. 3.

To execute our provided demo cases, the user can open our provided main files whose file names

Nr.	Code metadata description	Please fill in this column
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	
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

[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.

have the format of ”Example_main_*.m”, as shown in Fig. 3 yellow box. Then the user can execute the entire mfile by clicking the “Run” button. Each mfile can also be executed section by section by clicking the ”EDITOR >> RUN >> ▶” button (see Fig. 4A).

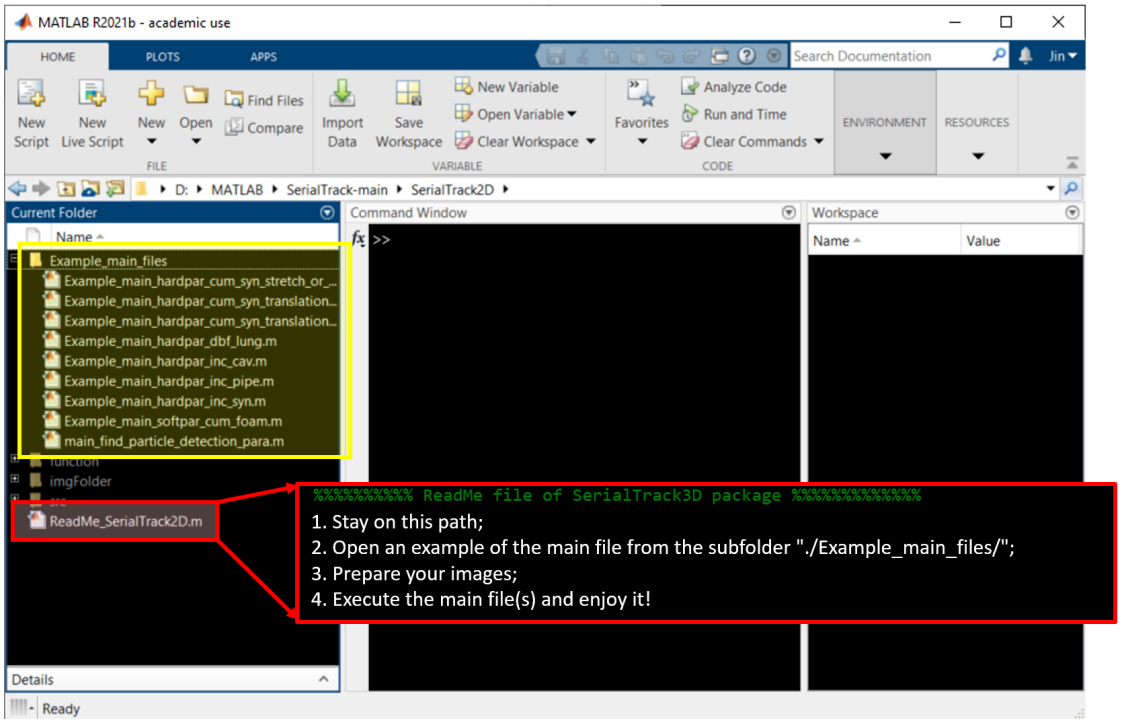

Figure 3: Example main files are stored in subfolder “./Example_main_files/”.

In this manual, we demonstrate the SerialTrack code package by testing 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 to track 2D rigid body motions including translations and rotations by executing the mfile “ `Example_main_hardpar_cum_syn_translation_or_rotation.m` ”, which is stored in subfolder “ `./SerialTrack2D/Example_main_files/` ”.

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

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% SerialTrack execute main file
% ==================================================
% Dimension:              2D
% Particle rigidity:      hard
% Tracking mode:          cumulative
% Syn or Exp:             syn
```

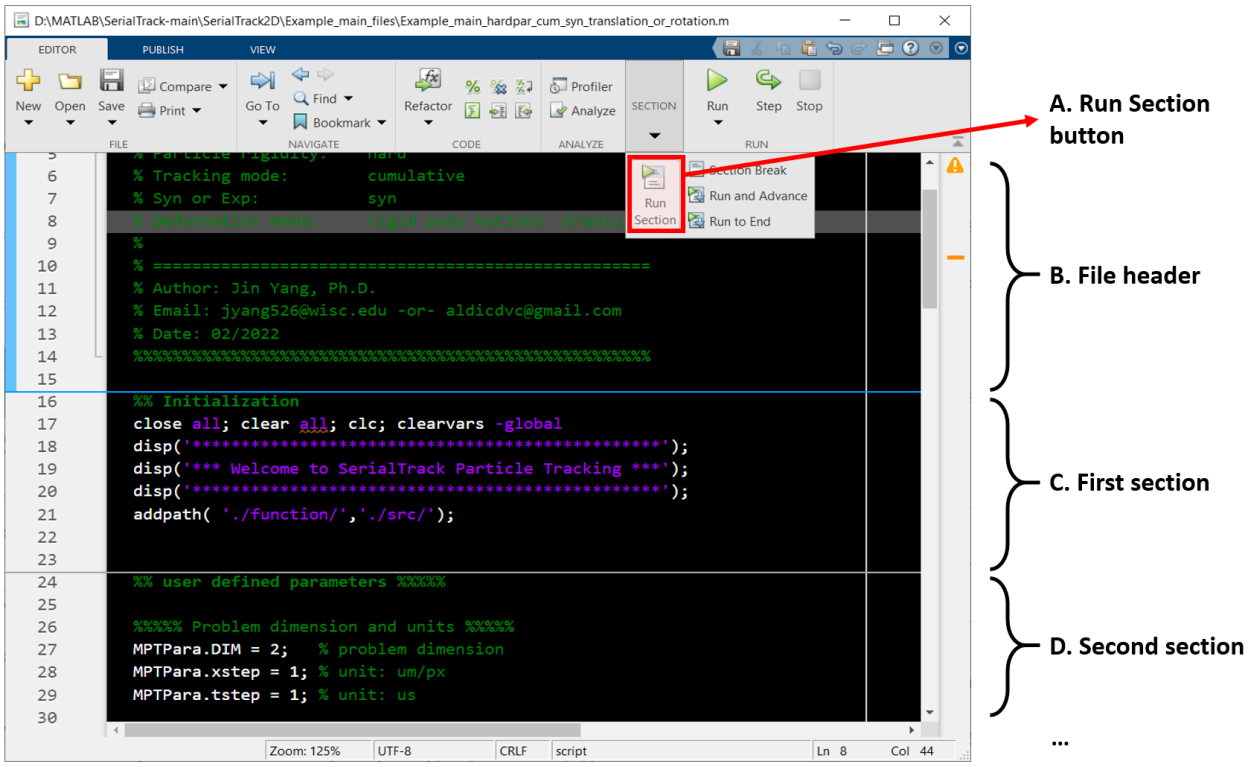

Figure 4: The screenshot of "Example_main_hardpar_cum_syn_translation_or_rotation.m" mfile. (A) The button to execute the selected code section. (B-D) File header, first and second sections of one typical main file.

```
% 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 as several sections (see Fig. 4C-D). Next, we will introduce these sections in Sections 3.1.1-3.1.8.

3.1.1 Initialization

The first section is to clear the MATLAB environment and add relevant subfolders (i.e., subfolders './functions/', './src/') to the MATLAB working path, 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

The parameter " `MPTPara` " is defined by the user based on the problem and the user chosen tracking mode. Since it is a 2D problem in this example, we define " `DIM` " as " `2` ". Here we define both " `MPTPara.xstep` " and " `MPTPara.tstep` " as " `1` ", which means that we choose "pixel" as our spatial unit and "1 (/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
```

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

```
%%%%% Tracking mode %%%%%
MPTPara.mode = 'cum'; % {'inc': incremental mode;
                      %  'cum': 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 MATLAB code, we 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 parameters

Next, we define particle detection parameters. In SerialTrack implementation, we leverage the state-of-the-art particle detection methods where the minimum size of the particles that can be effectively detected are at least 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 [5, 6]. In this example, recommended particle detection parameters are summarized as follows and their physical explanations are commented at the right.

```
%%%%% 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 radius of a single particle
BeadPara.maxSize = 20;             % Maximum radius of a single particle
BeadPara.winSize = [5, 5];         % By default
BeadPara.dccd = [1,1];             % By default
BeadPara.abc = [1,1];              % By default
BeadPara.forloop = 1;              % By default
BeadPara.randNoise = 1e-7;         % By default
BeadPara.PSF = [];                 % PSF function; Example: PSF = fspecial(
                                   % 'disk',BeadPara.beadSize-1); %Disk blur
BeadPara.distMissing = 2;          % Distance threshold to check whether
                                   % particle has a match or not
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, " `minSize` " and " `maxSize` " are the estimated lower and upper bounds of the particle radius.

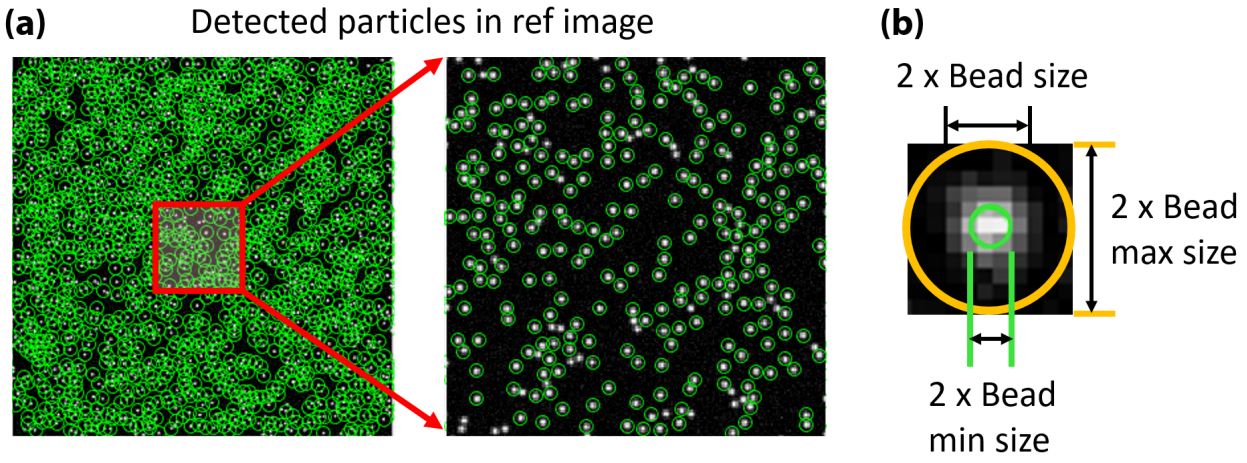

Figure 5: Particle detection results. (a) Detected particles in the reference image. (b) One typical detected particle with its detection parameters: beadSize, minSize, and maxSize.

3.1.4 Particle linking parameters

Besides particle detection parameters, we also need to define particle linking parameters where ***topology-based scale and rotation-invariant particle descriptors*** are further automatically generated. As shown in Fig. 6, we summarize a diagram outlining the feature descriptor generation process.

For each individual particle, the relative position between the k nearest neighbor particles and the selected particle is encoded into a complete particle descriptor consisting of two feature vectors.

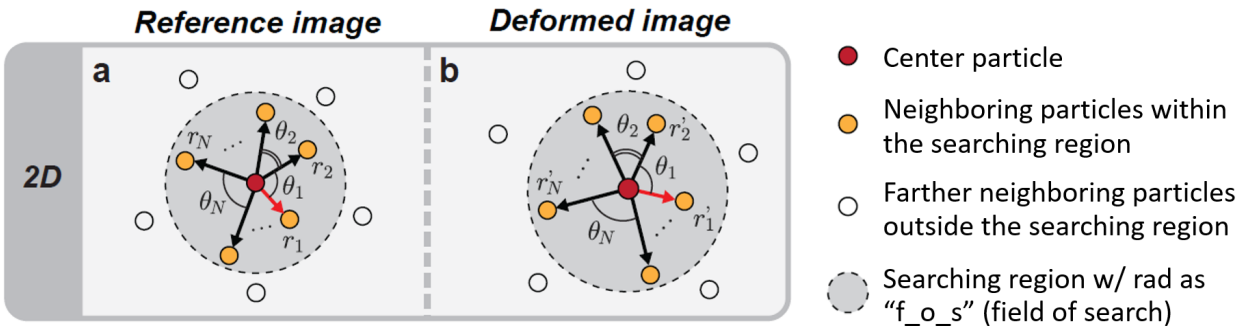

Figure 6: A diagram outlining the descriptor generation process. (a) The *k* radii and angles to nearest neighbor particles within the search distance ("f_o_s") for each particle. (b) The same computation is performed in the deformed image. Simultaneously minimizing Euclidean distance for angular and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

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 is the same with the nearest neighbor search method. However, if both " `MPTPara.n_neighborsMax` " and " `MPTPara.n_neighborsMin` " equal a same integer number greater than one, SerialTrack will obtain the same result with 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 TPT method and iteratively converge to the nearest neighbor search method. During this process, deformed images are also iteratively warped based on temporary tracked deformation fields.

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

```
%% SerialTrack particle tracking

%%%%% Multiple particle tracking (MPT) Parameter %%%%%
MPTPara.f_o_s = Inf;                % Size of search 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
MPTPara.usePrevResults = 0;     % Whether use previous results or not:
                                % 0-no; 1-yes;
```

3.1.5 Merging trajectory segments

This part is to merge tracked particle trajectory segments, which mostly considers 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. The explanations of these parameters are listed at 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
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
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, which including '***inc***': incremental tracking mode; '***cum***': 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,'cum')==1
    run_Serial_MPT_2D_hardpar_cum;
elseif strcmp(MPTPara.mode,'dbf')==1
    run_Serial_MPT_2D_hardpar_dbf;
end
```

3.1.7 Code execution results

Executing the whole mfile " `Example_main_hardpar_cum_syn_translation_or_rotation.m` ", the user will be asked to load both 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: cum
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
```

The code will require the user to define the region of interest on the reference image. Here we assume both reference and all deformed images have 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. MATLAB command screen 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 window will display the tracking results as shown below:

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

====== 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 ------
```

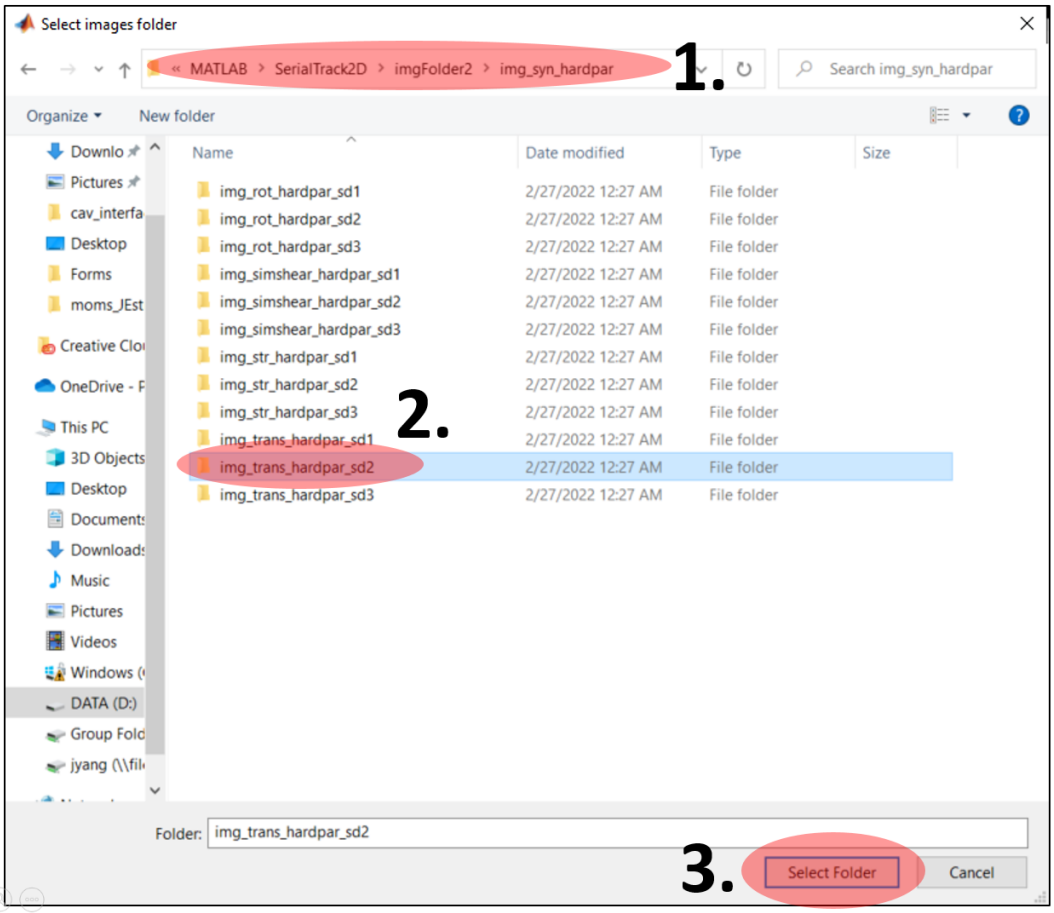

Figure 7: Images can be loaded by manually selecting an image folder where all frames are stored. Steps 1-2: To select the image folder storing images; Step 3: To click the “Select Folder” button to finish this step.

```
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.
```

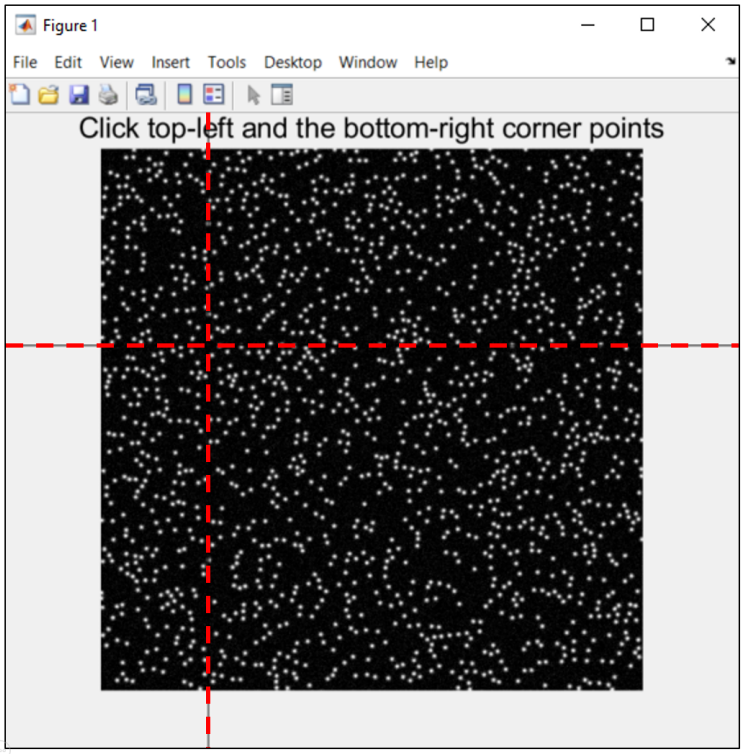

Figure 8: A rectangular region of interest (ROI) can be defined by manually clicking a top-left point and a bottom-right corner point

```
33
34 ====== Frame #3 ======
35 ...
36 (Results of other frames can be found by executing the code.)
37 ...
38
39 %%%%% Calculate incremental tracking ratio %%%%%
40
41 %%%%% SerialTrack hard particle tracking: Done! %%%%%
42
43 %%%%% Plot tracked cumulative deformations %%%%%
44
45 %%%%% Plot tracked trajectories %%%%%
46
47 Press "Ctrl + C" and modify codes below to plot interpolated displacements
       and strains on a uniform grid mesh
```

The tracking results of 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 : 10 : 180] degrees) are summarized in Fig. 9 and Fig. 10, respectively.

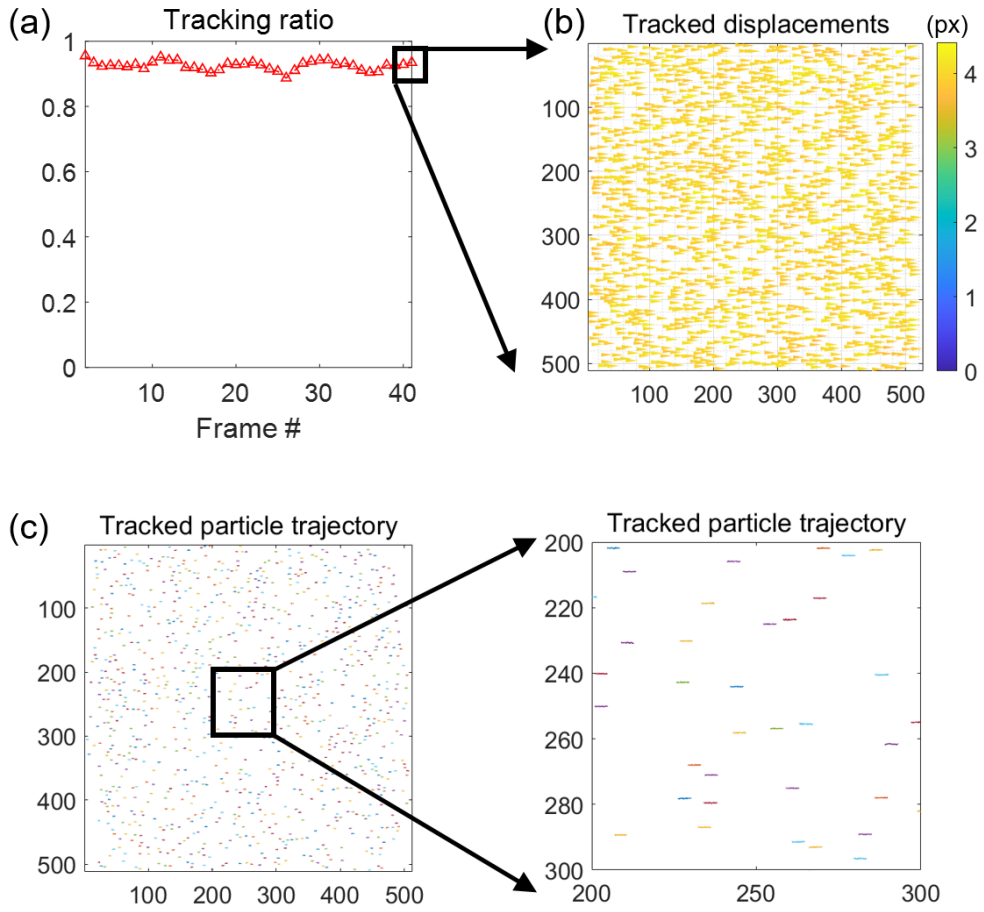

Figure 9: Tracked 2D rigid body translations. Frames 2-41 have rigid horizontal translations of $[0.1 : 0.1 : 4]$ pixels. (a) Final tracking ratios for subsequent frames. (b) The cone plot of the tracked displacement field of the 41-th frame. (c) The trajectory plot of tracked particles, where a window of [200,300] pixel $\times$[200,300] pixel is further zoomed in at the right.

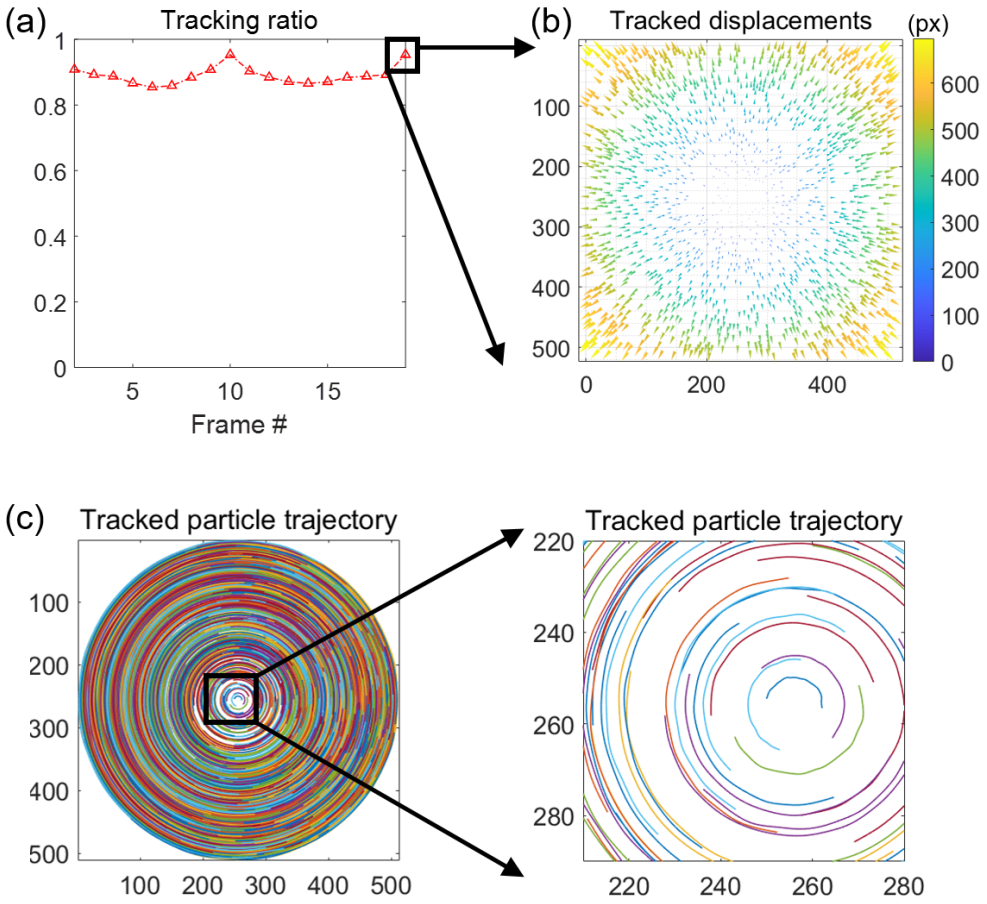

Figure 10: Tracked 2D rigid body rotations. Frames #2-#19 have rotation angles of [10 : 10 : 180] degrees. (a) Final tracking ratio for subsequent frames. (b) The cone plot of the tracked displacement field of the 18-th frame. (c) The trajectory plot of tracked particles, where a window of [210,280]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

3.1.8 Post-processing

Recall that SerialTrack direct tracked particle motions are scattered results. Here, we also provide codes to interpolate these scattered results to a regularly gridded mesh.

The user needs to assign the frame # to perform the interpolation.

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

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

For example, results of the second frame (“ `ImgSeqNum = 2` ”) in Fig. 10(a) are further interpolated based on a regularly gridded mesh and shown in Fig. 11.

```
%%%%% Previously tracked displacement field %%%%%
resultDispCurr = resultDisp{ImgSeqNum-1};
resultDefGradCurr = resultDefGrad{ImgSeqNum-1};
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
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 world
    units
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 %%%%%
[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 stretches (stretch ratios are from [100 : 5 : 300] %) and simple shears (shear angle γ from 0° to 45° in $\tan(\gamma)$ increments of 0.05). The user can execute the mfile " `Example_main_hardpar_cum_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 tracked displacement fields of uniaxial stretches and simple shears in Fig. 12 and Fig. 13, respectively.

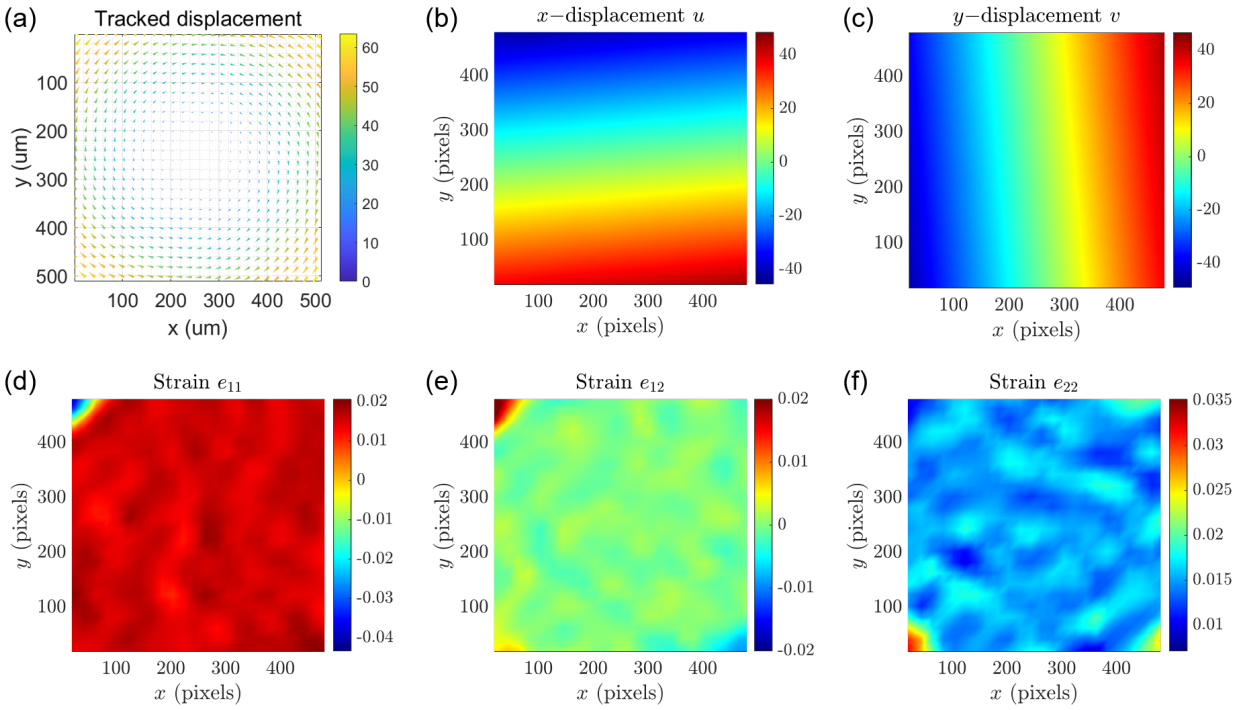

Figure 11: Results of the second frame (“ `ImgSeqNum = 2` ”) in Fig. 10(a) are further interpolated onto a regularly gridded mesh. (a) The cone plot of the interpolated displacement field. (b-c) Interpolated x and y displacement components. (d-f) Interpolated e_{xx}, e_{xy}, and e_{yy} strain fields.

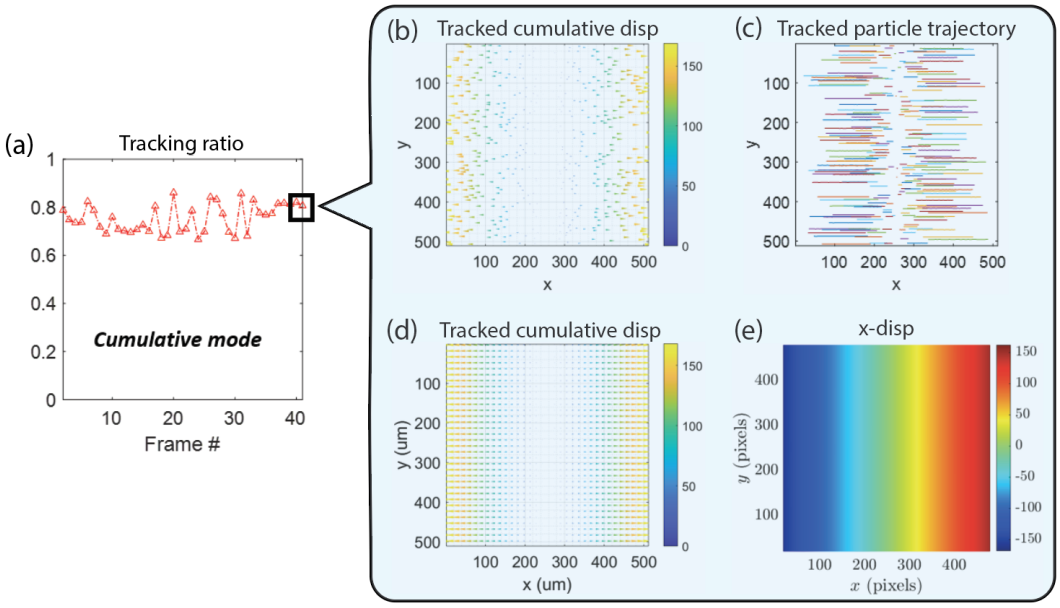

Figure 12: Tracked 2D uniaxial stretches. Stretch ratios are from [100 : 5 : 300] %. (a) Final tracking ratios. (b) The cone plot of tracked cumulative displacement field. (c) Plot of tracked particle trajectory. (d) The interpolated tracked cumulative displacement on a regularly gridded mesh. (e) The interpolated tracked x-displacement field.

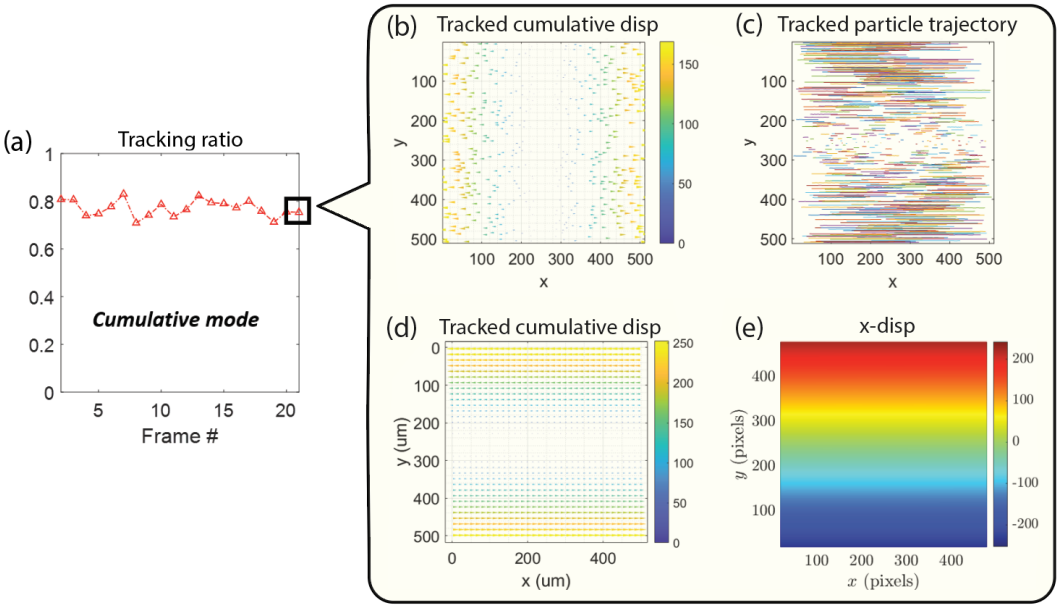

Figure 13: Tracked 2D simple shears. Shear angle γ from 0° to 45° in tan(γ) increments of 0.05. (a) Final tracking ratios. (b) The cone plot of tracked cumulative displacement field. (c) Plot of tracked particle trajectory. (d) The interpolated tracked cumulative displacement on a regularly gridded mesh. (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 introduce how to track a 2D complex geometry assisted with a binary mask file. The used data set from an experimental test of a pipe flow can be obtained from [6]. To measure 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 path (see Fig. 14). This mat file needs to be a saved binary matrix (or a/multiple binary mask file(s) can be uploaded) whose size is the same with the reference image. **In this binary matrix, "0" means the background and "1" means the sample area. Only particles on the sample 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 utilized mask file.**

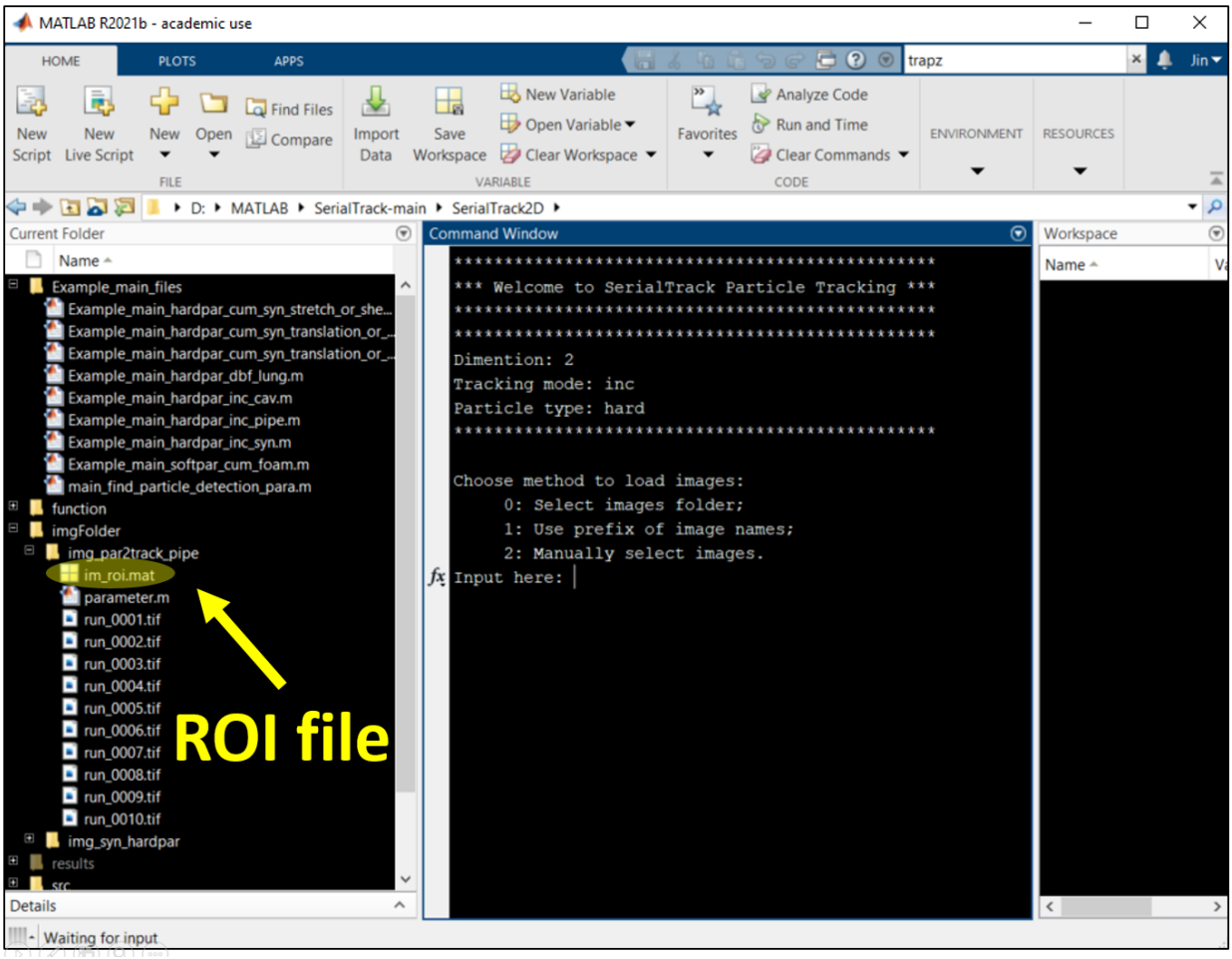

Figure 14: A prior ROI mat file needs to be prepared and stored to track motions/deformations of a complex geometry.

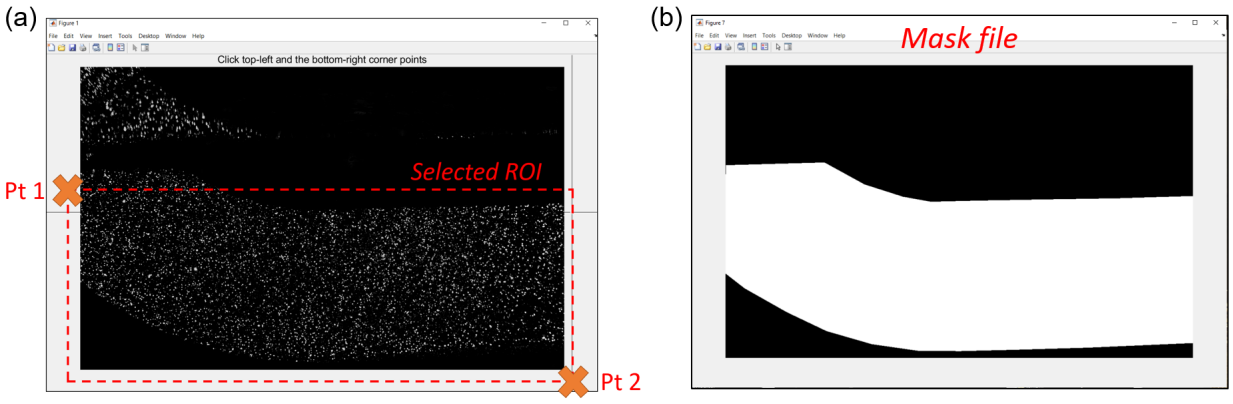

Figure 15: (a) A ROI can be defined by the user by manually clicking the top-left and bottom-right corner points. (b) The utilized mask file in this example.

3.3.1 User defined parameters

Again, the user can customize the particle detection and linking parameters. 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 incremental tracking mode (`MPTPara.mode = 'inc'`) since there can be large deformations by comparing the first frame and subsequent frames. In addition, we care more about the incremental deformation fields (or the velocity fields) rather than the cumulative displacement fields of flow through a bent pipe. Since there are almost no shape distortions for moving particles, we define the particle rigidity as "hard" (" `MPTPara.parType = 'hard'` "). All these details correspond to following lines:

```
%%%%% Code mode %%%%%
MPTPara.mode = 'inc'; % {'inc': incremental mode;
                      %  'cum': 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 parameter " `MaskFileLoadingMode = 3;` " and provide its storing path.

```
%%%%% 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

We define 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 radius of a single particle
BeadPara.maxSize = 20;          % Maximum radius of a single particle
BeadPara.winSize = [5, 5];      % By default
BeadPara.dccd = [1,1];          % By default
BeadPara.abc = [1,1];           % By default
BeadPara.forloop = 1;           % By default
BeadPara.randNoise = 1e-7;      % By default
BeadPara.PSF = [];              % PSF function; Example: PSF = fspecial(
                                % 'disk',BeadPara.beadSize-1); %Disk blur
BeadPara.distMissing = 2;       % Distance threshold to check whether
                                % particle has a match or not
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|)
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
MPTPara.usePrevResults = 0;       % Whether use previous results or not:
                                  % 0-no; 1-yes;
```

3.3.4 Merging trajectory segments

We merge tracked trajectory segments using 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 the distance threshold (`distThres`) to connect split trajectory segments. Because there may exist large errors for extrapolations, we will not merge segments if the frame numbers of their two ends are greater than the user defined parameter `'maxGapTrajSeqLength'` – which is defined as the maximum frame number gap between connected trajectory segments.

```
%%%% Postprocessing: merge trajectory segments %%%%%
distThres = 1;            % distance threshold to connect split
                          % trajectory segments
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 extrapolated
maxGapTrajSeqLength = 0; % the max frame # gap between connected
                          % trajectory segments
```

3.3.5 Tracked results

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

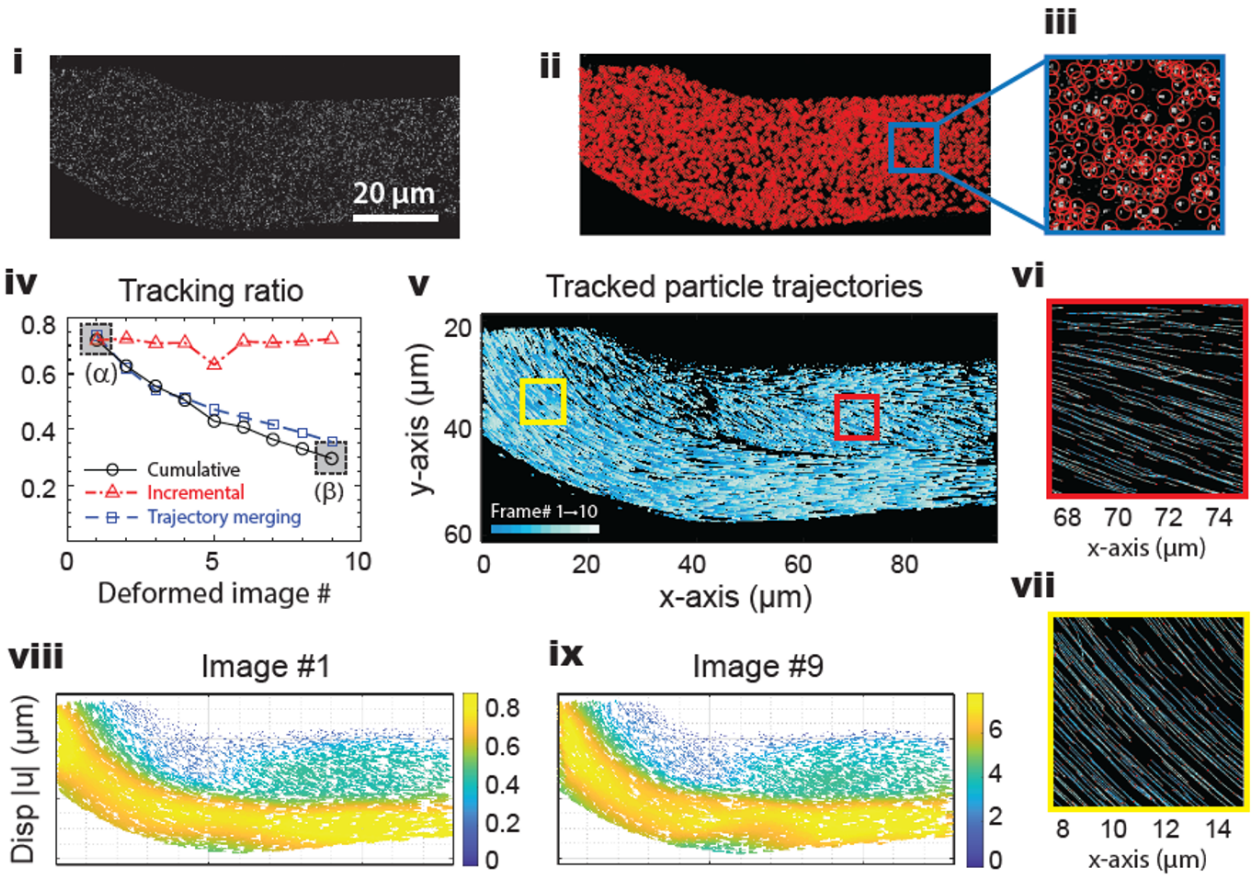

Figure 16: The results of the example of flow through a bent pipe [6]. (i) One typical frame in the time-resolved image sequence. (ii-iii) Detected single particle centroids using the Laplacian of Gaussian filtering technique are circled in red. (iv) Particle tracking ratios. (v-vii) Tracked trajectories. (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 “double frame” mode, two frames are taken under 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]). Here we take the flow inside human lungs as an example. The original data set is accessible from Janke, et al. [6]. The experimental field of view is illustrated in Fig. 18(a) using a green box. As shown in Fig. 17(a), all the image pairs are stored in a folder on the MATLAB working path. Each image pair is taken with a temporal delay △t and the relative motion between these two frames can be used to calculate real time velocity fields, as shown in Fig. 17(b).

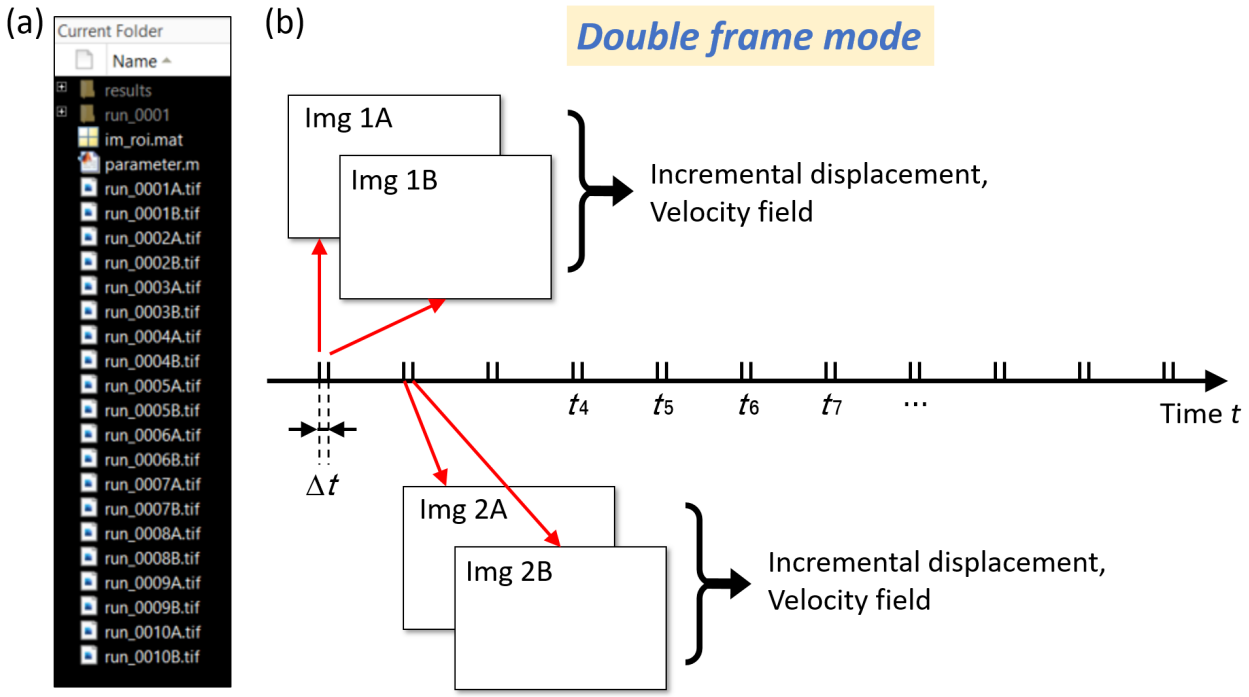

Figure 17: Double frame tracking mode. (a) All the image pairs are stored in a folder on the MATLAB working path. (b) In double frame mode, two frames are taken under every single exposure with a temporal delay and each odd number frame is compared to its subsequent even number frame (more details can 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 zoomed in to show that randomly scattered particles can be precisely detected using 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
BeadPara.minSize = 2;           % Minimum radius of a single particle
BeadPara.maxSize = 200;         % Maximum radius of a single particle
BeadPara.winSize = [5, 5];      % By default
BeadPara.dccd = [1,1];          % By default
BeadPara.abc = [1,1];           % By default
```

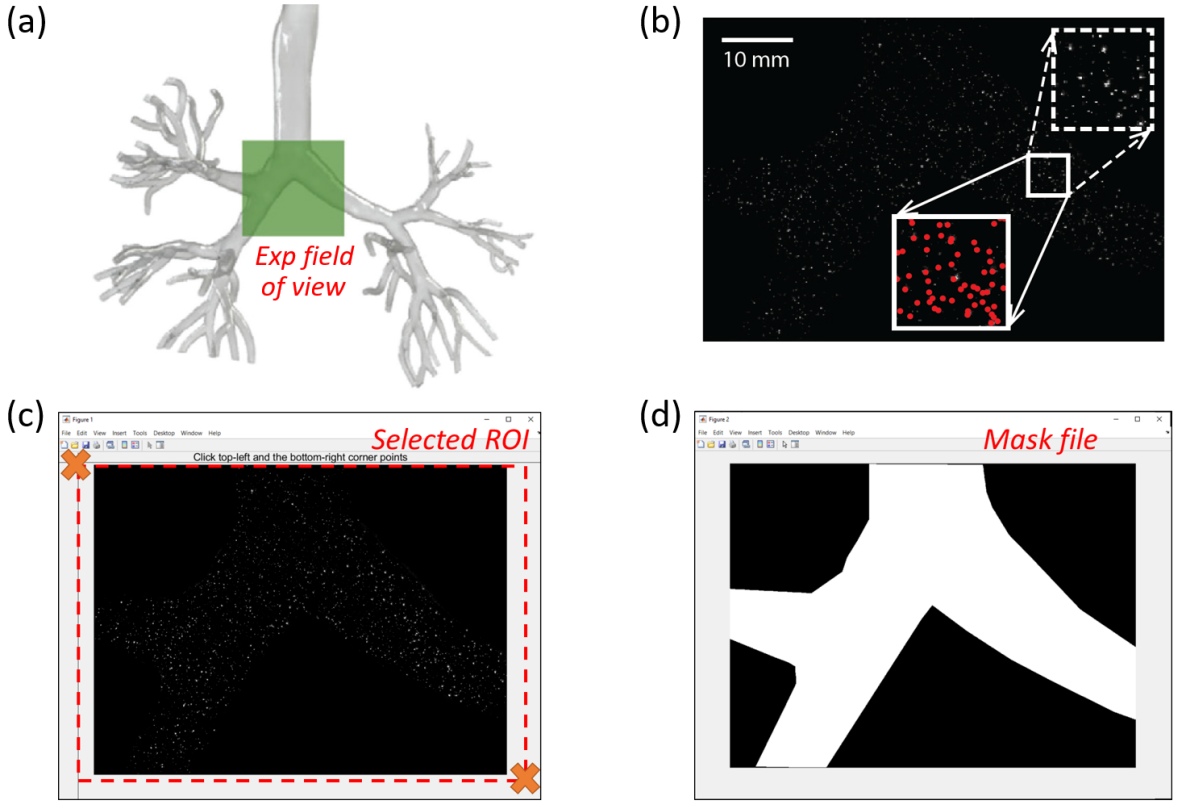

Figure 18: (a) The field of view in the experimental set up to measure the flow field inside the human lungs. (b) One typical captured fame, where a small white square box is further zoomed in to show that centroids of randomly scattered particles can be precisely detected. (c) The defined region of interest (ROI) by manually clicking a top-left and a bottom-right points. (d) An image binary mask file can be further applied to define an complex geometry.

```
BeadPara.forloop = 1;                % By default
BeadPara.randNoise = 1e-7;           % By default
BeadPara.PSF = [];                   % PSF function;
                                     % Example: PSF = fspecial('disk', ...
                                     % BeadPara.beadSize-1); % Disk blur
BeadPara.distMissing = 2;            % Distance threshold to check whether
                                     % particle has a match or not
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 a top-left and a right-bottom corner points (see Fig. 18(c)) and an image binary mask file (see Fig. 18(d)) can also be applied by providing its path to define a complex geometry. **The final tracking area is the intersection of the manually defined ROI and the utilized 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 tracked results of the flow inside human lungs. For each image pair in double frame mode, the 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 is shown in (d) and (e), respectively.

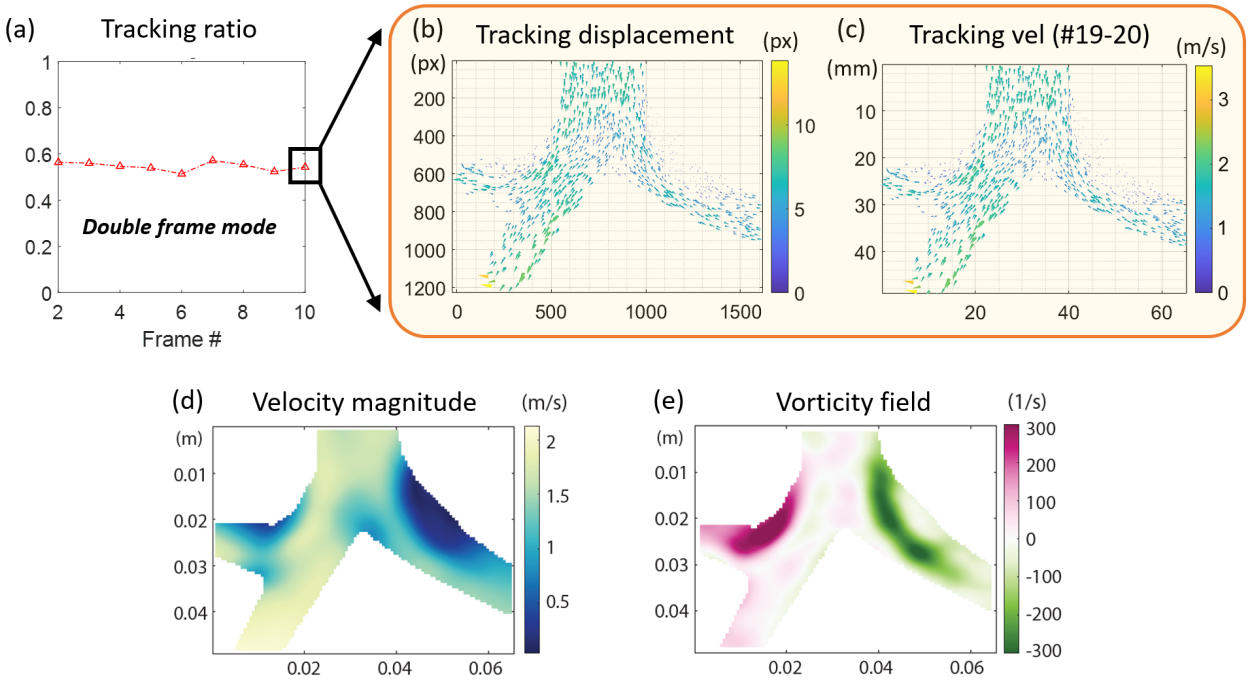

Figure 19: Summary of tracked results of the flow inside the human lungs. (a) The final, tracking ratios. (b-c) An example of the tracked incremental displacement and corresponding velocity fields. (d-e) The averaged velocity and calculated vorticity fields using all ten image pairs.

3.5 2D “Soft particles” shape distortions: foam compression test

In 2D cases, particles can be the user customizedly printed or painted on the sample surface [10]. Therefore, these particles may have significant shape distortions due to corresponding sample deformations. For example, a circular speckle dot on a 2D sample surface may deform into an ellipse during a uniaxial tension/compression test, which might degrade the particle detection and decrease the tracking accuracy. Here, we define particles as “hard” particles if they are effectively rigid and their shapes don’t change throughout the experiment. Alternatively, we call particles to be “soft” if we assume their shape change. Here, we make assumption that their shape distortion coincides with local deformation gradients, as shown in Fig. 20(b).

We optionally consider the effect of particle shape distortion in SerialTrack. Different from “hard” particle tracking, in the “soft” tracking algorithm, particle centroid localtions will be re-detected during each iteration step using warped images in Algorithm 2 iterations (see our original paper Algorithm 2).

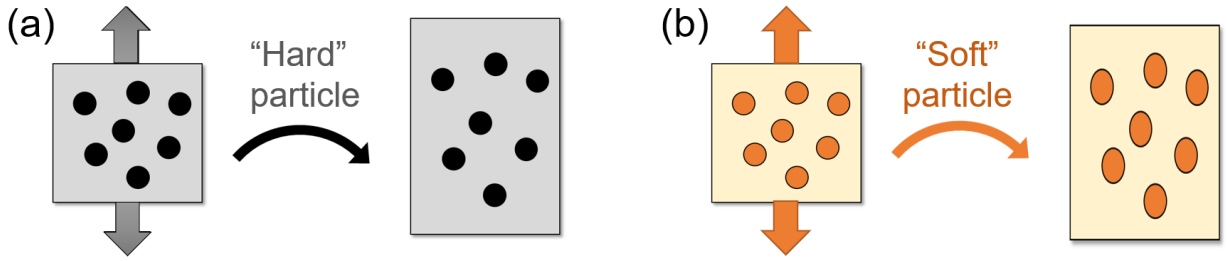

Figure 20: (a) We consider particles “hard” if the shape is effectively rigid and invariant throughout the experiment. (b) Alternatively, we consider particles to be “soft” if particle shape distorts and we assume their shape distortion coincides with local deformation gradients.

The user can open and execute the mfile “Example_main_softpar_cum_foam.m” which is stored in the subfolder “Example_main_files”. Similar to Section 3.1, the mfile header describes basic information of 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 `User defined parameters` section, we define problem variables.

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

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

%%%%% Code mode %%%%%
MPTPara.mode = 'cum'; % {'inc': incremental mode;
                      %  'cum': 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 parameters

Particle detection parameters are defined as

```
%%%%% Particle detection parameters %%%%%
%%%%%%%%%% Elastomeric open-cell foam %%%%%%%%%%%%
%%%%% Bead Parameter %%%%%
BeadPara.thres = 0.3;           % Threshold for detecting particles
BeadPara.beadSize = 0;          % Estimated radius of a single particle
BeadPara.minSize = 3;           % Minimum radius of a single particle
BeadPara.maxSize = 200;         % Maximum radius of a single particle
BeadPara.winSize = [5, 5];      % By default
BeadPara.dccd = [1,1];          % By default
BeadPara.abc = [1,1];           % By default
BeadPara.forloop = 1;           % By default
BeadPara.randNoise = 1e-7;      % By default
BeadPara.PSF = [];              % PSF function;
                                % Example: PSF = fspecial('disk', ...
                                % BeadPara.beadSize-1 ); % Disk blur
BeadPara.distMissing = 10;      % Distance threshold to check whether
                                % particle has a match or not
```

```
BeadPara.color = 'black';       % Bead color: 'white' -or- 'black'
```

3.5.3 Particle linking parameters

All other 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|)
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
MPTPara.usePrevResults = 1;       % Use previous results? 0-no; 1-yes;


%%%% Postprocessing: merge trajectory segments %%%%%
distThres = 1; % distance threshold to connect split trajectory segments
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
```

3.5.4 Executing the SerialTrack particle tracking code

So far, we only implement the cumulative tracking mode to track soft particles since particle shape distortion only appears if there are also significant cumulative strain distortions. We will start the SerialTrack particle tracking after executing following lines:

```
%%%%% Execute SerialTrack particle tracking %%%%%
if strcmp(MPTPara.mode,'inc')==1
    disp('Not implemented yet.');
    % run_Serial_MPT_2D_softpar_inc;
elseif strcmp(MPTPara.mode,'cum')==1
    run_Serial_MPT_2D_softpar_cum;
elseif strcmp(MPTPara.mode,'dbf')==1
```

```
    disp('Not implemented yet.');
    % run_Serial_MPT_2D_hardpar_dbf;
end
```

After executing the mfile, the user will be requested to select the image folder to load images, see Fig. 21. This step is same as before. Here we test 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].

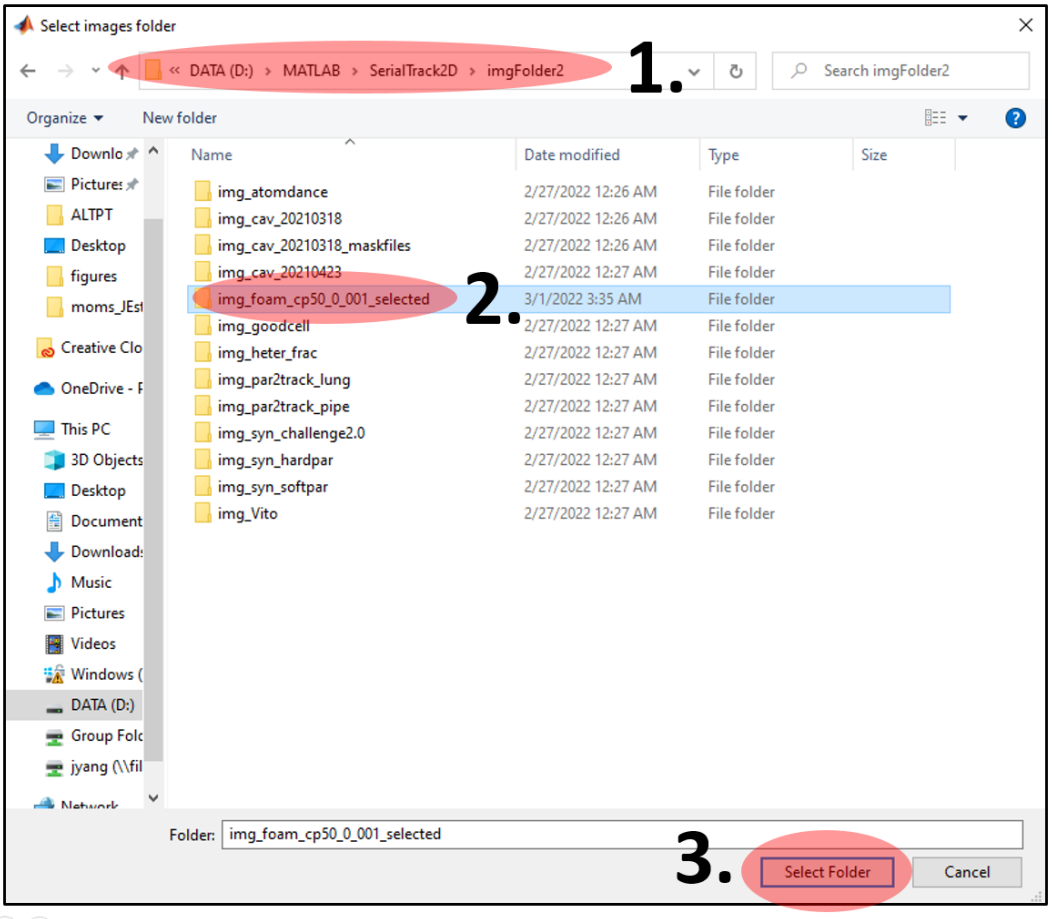

Figure 21: Images can be loaded by manually selecting an image folder where all frames are stored.

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

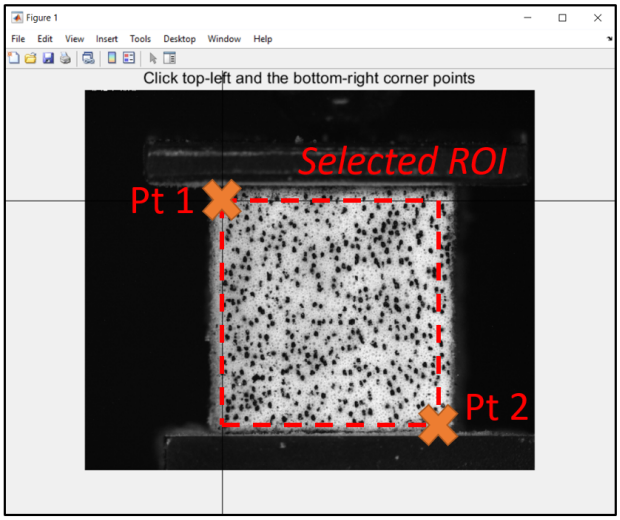

Figure 22: A ROI can be manually defined by clicking the top-left and bottom-right corner points.

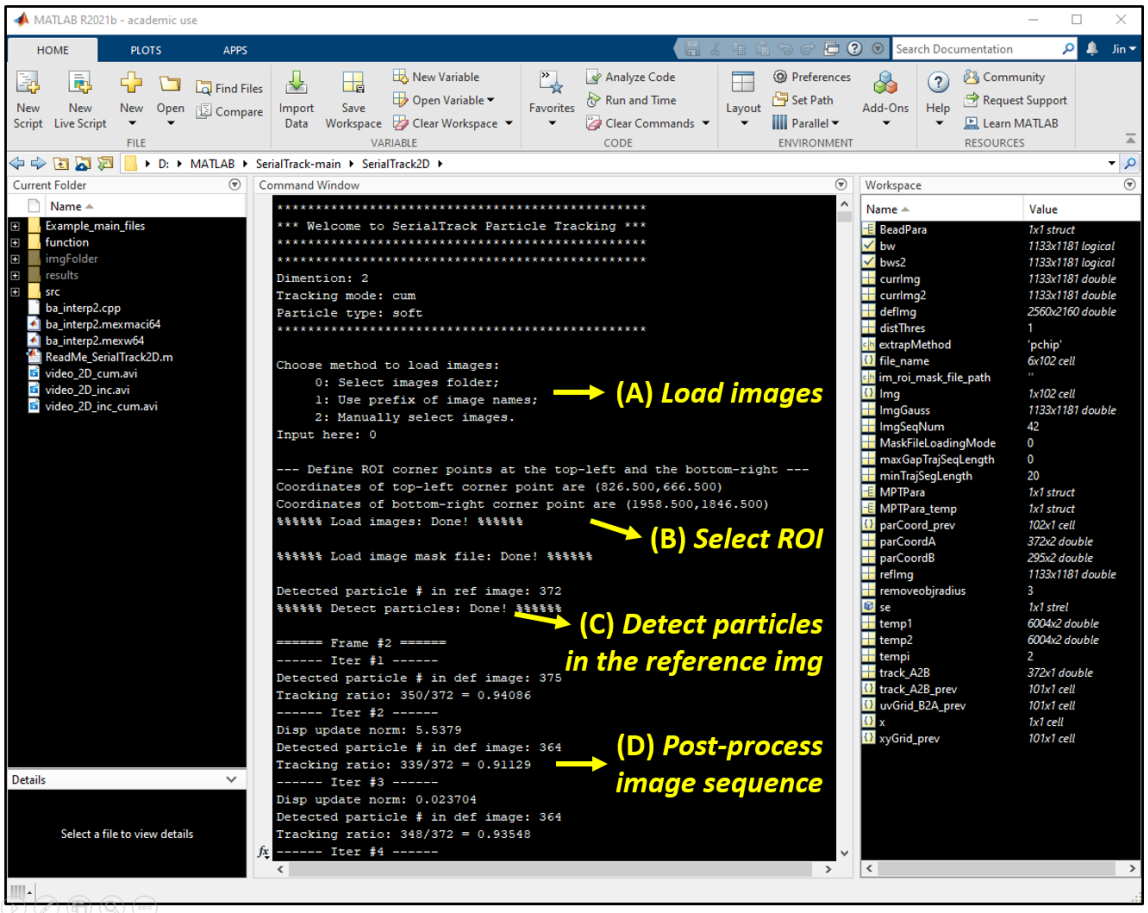

Figure 23: The screen shot of the MATLAB Command Window during code execution.

4 SerialTrack 3D Examples

Similarly to 2D cases, the users can detect 3D bead centroids and track their motions by comparing deformed volumetric images to the reference volumetric image. Several examples of MATLAB execution main files are stored in the subfolder "./Example_main_files" (see Fig. 24(A)). Compared to 2D cases, there are two changes in 3D cases. 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 build each individual particle's 3D topology-based feature descriptors instead of 2D descriptors (see main manuscript Figure 2(c-d)).

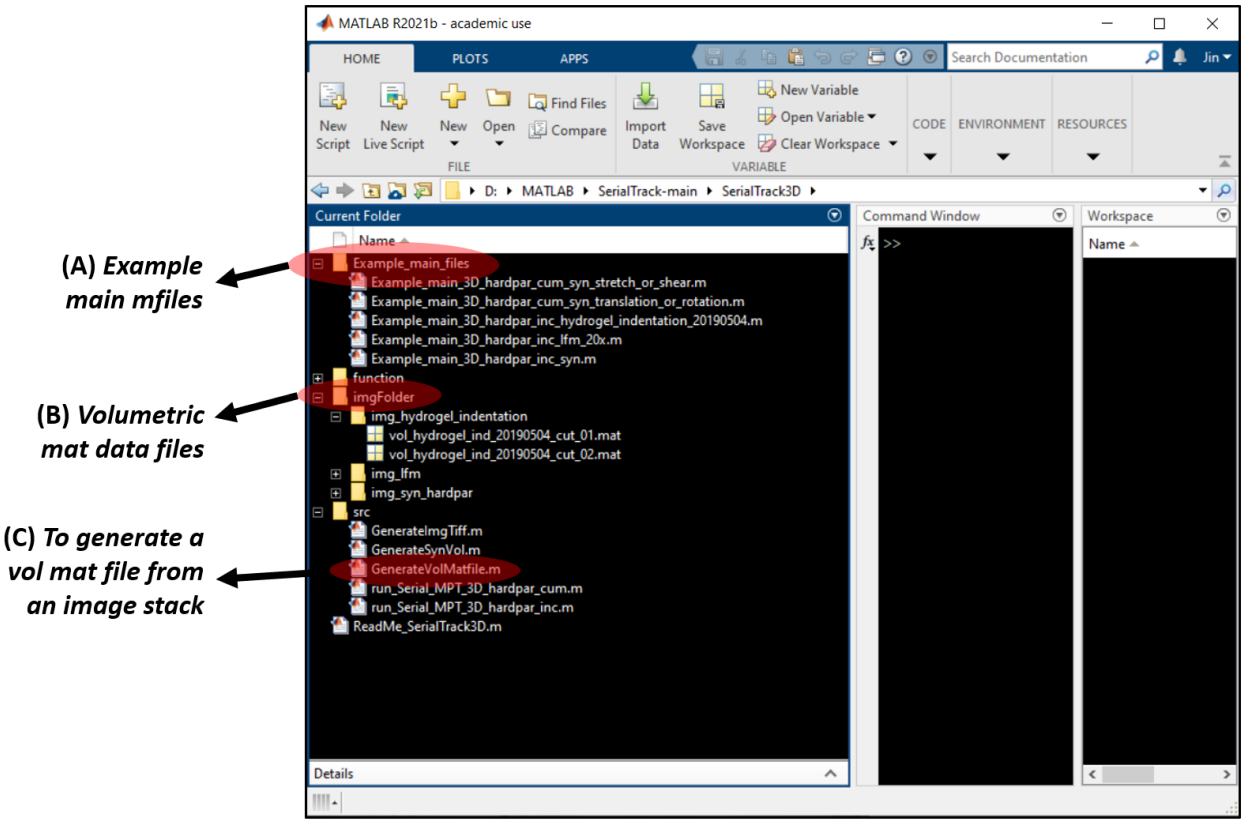

Figure 24

4.1 Transforming image stacks to MATLAB mat files

The user can store their image stack in a subfolder, as shown in Fig. 25. Next, the user needs to modify several `TODO` places in the "GenerateVolMatfile.m" to transform image stacks to MATLAB volumetric matrices and store them as mat files. All these "TODO" steps will be explained as follows.

First, the user needs to add all the stored image stacks to the MATLAB working path and in the same folder, as shown in Fig. 25.

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

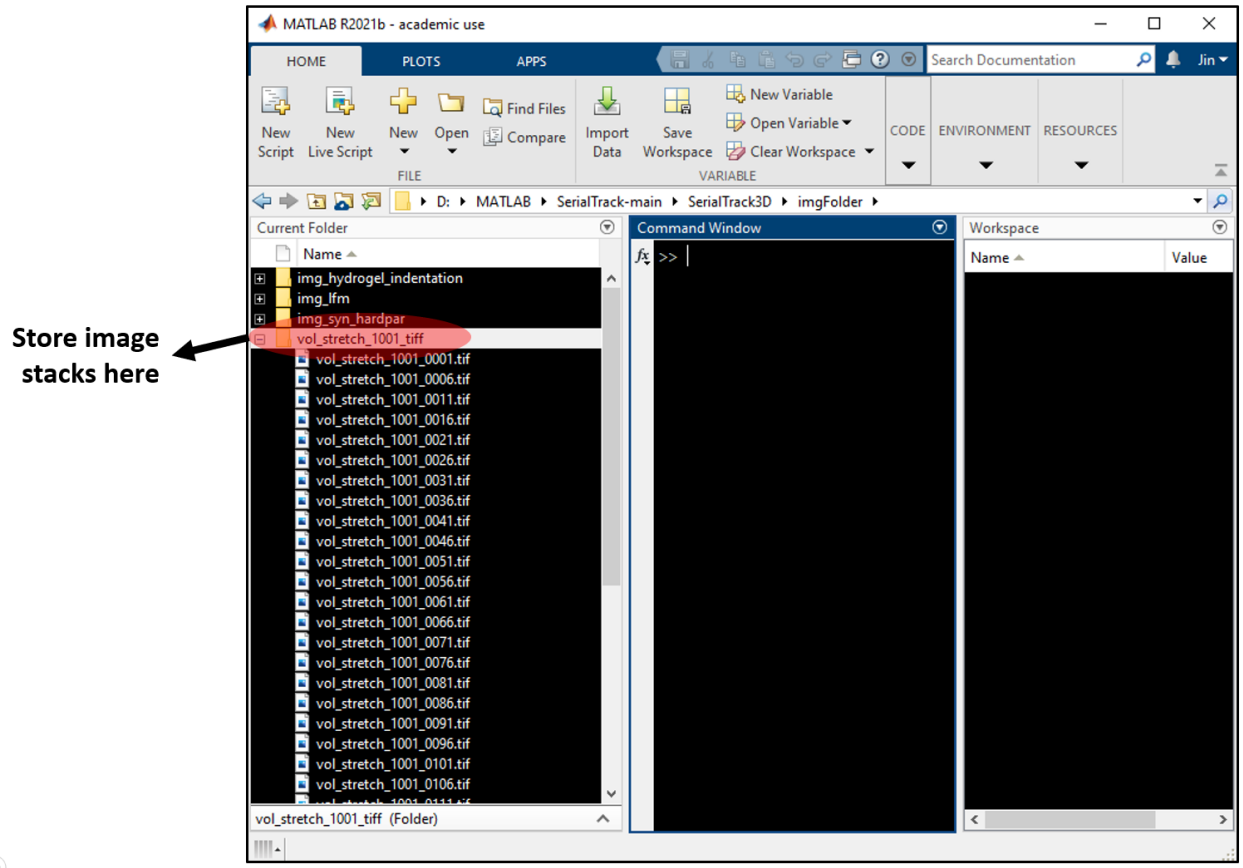

Figure 25: Image stacks are stored in the same folder.

We use “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 also 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 find the bit depth of the image stack and modify the following line correspondingly.

```
% 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]));
```

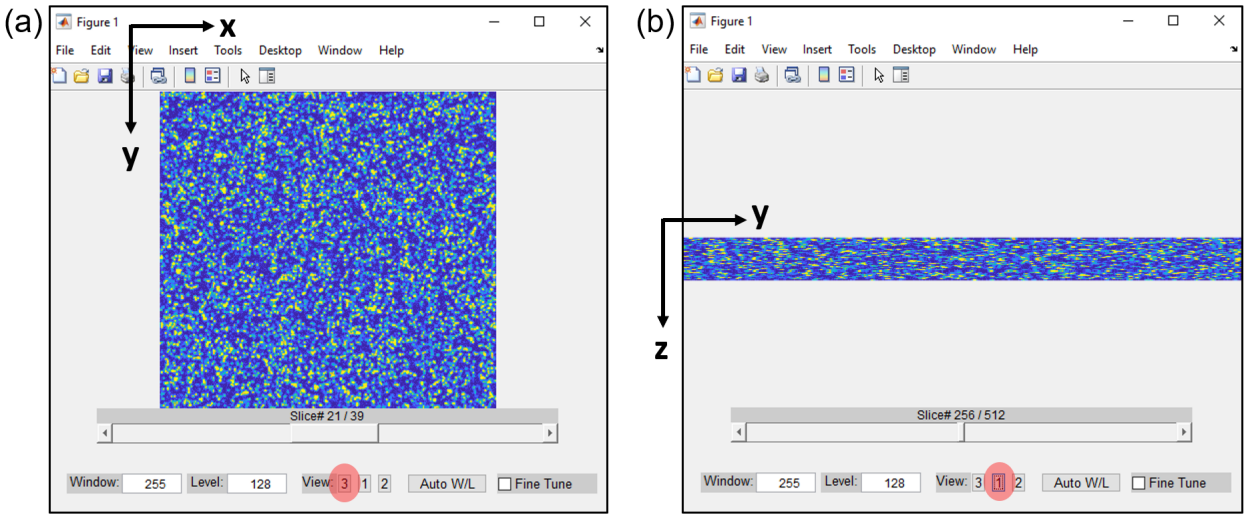

Figure 26: Generated volumetric matrix can be visualized using the “imagesc3” function.

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 “GenerateSynVol.m”. This file can simulate four different types (“ `DefType` ”) of uniform deformations: **(a) rigid body translations**, **(b) rigid body rotations around z-axis**, **(c) uniaxial stretches**, and **(d) simple shears**, 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 synthesized data sets 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/”. These lines need to modify correspondingly.

```
%%%%% SerialTrack path %%%%%
SerialTrackPath = 'D:\MATLAB\SerialTrack3D'; % TODO: modify the path

%%%%% 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 also provide a real 3D volumetric experimental example to demonstrate our SerialTrack-3D code. The original data set can be downloaded from [4]. This is a 3D hydrogel indentation experiment where a steel sphere on the top indents the bottom hydrogel using its gravity.

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

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 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

%%%%% Code mode %%%%%
MPTPara.mode = 'inc'; % {'inc': incremental mode;
                      %  'cum': 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 = 'D:\MATLAB\SerialTrack-main\SerialTrack3D'; % TODO:
    modify the path

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

4.3.2 Particle detection parameters

```
%%%%% Bead detection method %%%%%
BeadPara.detectionMethod = 2;  % {1-TPT code; 2-regionprops}


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


%%%%% Particle detection parameters %%%%%
%%%%% Bead Parameter %%%%%
BeadPara.thres = 0.1;           % Threshold for detecting particles
BeadPara.beadSize = 3;          % Estimated radius of a single particle
BeadPara.minSize = 3;           % Minimum radius of a single particle
BeadPara.maxSize = 20;          % Maximum radius of a single particle
BeadPara.winSize = [5,5,5];     % By default
BeadPara.dccd = [1,1,1];        % By default
BeadPara.abc = [1,1,1];         % By default
BeadPara.forloop = 1;           % By default
BeadPara.randNoise = 1e-7;      % By default
BeadPara.PSF = [];              % PSF function; Example: PSF = fspecial('
                                % disk', BeadPara.beadSize-1 );
                                % Disk blur
BeadPara.distMissing = 5;       % Distance threshold to check whether
                                % particle has a match or not
BeadPara.color = 'white';       % By default
```

4.3.3 Particle linking parameters

As we mentioned before, we need to construct 3D feature descriptor for each individual particle. Analogous to the 2D descriptor, 3D radial distances (r), polar angles (θ), and azimuthal angles (ϕ) of the stored k neighboring particles are used to construct particle descriptors for each particle. To establish a coordinate system, $\{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\}$, for each particle, we define the first nearest neighbor

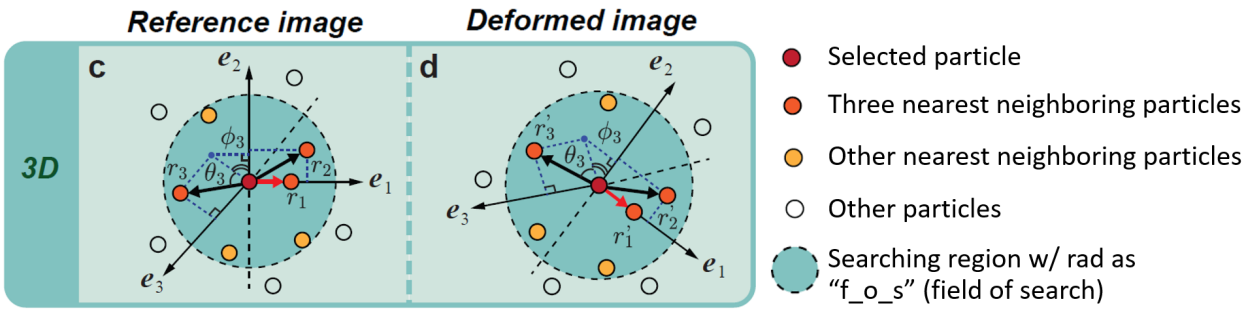

Figure 27: A diagram outlining the 3D descriptor generation process. (a) The *k* radii and angles to nearest neighbor particles within the search distance ("f_o_s") for each particle. (b) The same computation is performed in the deformed image. Simultaneously minimizing Euclidean distance for angular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

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 will be further normalized by the first nearest neighbor particle distance.

```
%%%%% Multiple particle tracking (MPT) Parameter %%%%%
MPTPara.f_o_s = 60;                % Size of search field: max(|u|,|v|,|w|)
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
MPTPara.usePrevResults = 0;        % Whether use previous results or not:
                                   % 0-no; 1-yes;
```

4.3.4 Merging trajectory segments

```
%%%% Postprocessing: merge trajectory segments %%%%%
distThres = 1;                % distance threshold to connect split
                              % trajectory segments
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
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,'cum')==1
    run_Serial_MPT_3D_hardpar_cum;
end
```

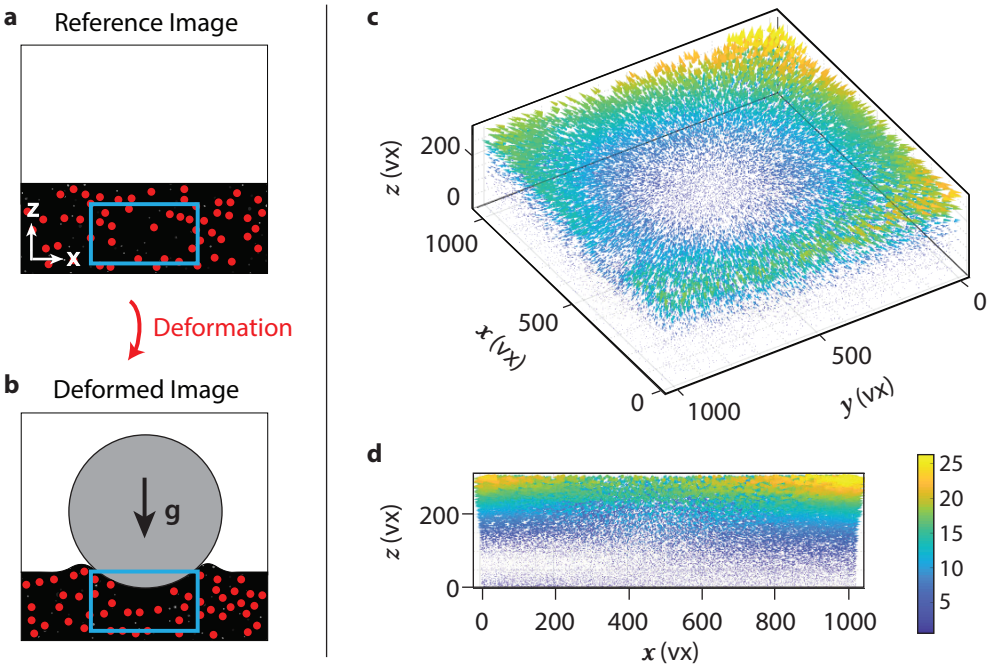

Figure 28: A 3D spherical hydrogel indentation example. (a-b) Sketch of the reference and deformed configurations of the spherical indentation experiment. (c-d) 3D cone plot and xz-plane projection of the tracked 3D deformation.

Acknowledgements

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 while A.K.L. held an NRC Research Associateship award with Aaron Forster at the National Institute of Standards and Technology.

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.