ObDiTo Manual

Ryan Sills*, Rutgers University, January 2022

*ryan.sills@rutgers.edu, please feel free to contact me with any questions or to report issues!

1. Purpose and Scope

The purpose of this manual is to detail how to use the Object-based Dislocation Tomography (ObDiTo) code, which was written to perform three-dimensional tomographic reconstructions from tilt series images obtained in the transmission electron microscope (TEM). For information on the algorithm and operating principles of ObDiTo, see:

R. B. Sills and D. L. Medlin, *Semi-automated, Object-Based Tomography of Dislocation Structures*, In Preparation, 2022.

When using ObDiTo, please reference this publication.

ObDiTo has three core code modules, which correlate with the three steps of dislocation object-based reconstruction:

`extract_lines` – takes a TEM image as input and extracts discrete dislocation lines with graphical input from users

`match_objects` – takes sets of dislocation lines previously extracted from TEM images and provides a graphical tool for users to "match" the same dislocation "object" across images

`reconstruct_3dlines` – takes sets of matched objects and performs a 3D reconstruction of the dislocation network formed by the objects

The final output of `reconstruct_3dlines` is a data structure which describes the 3D arrangement of the dislocation lines in terms of nodes and line segments. This data structure is comprised of a N x 3 `rn` array (N is the number of nodes) with the ith row as the [x, y, z] coordinates of node with ID i, and a M x 2 `links` array (M is the number of line segments) with the jth row as the node IDs to which the jth segment is connected. This is the same data structure used by the `ddlab` discrete dislocation dynamics code written in MATLAB by Wei Cai[1].

In addition to the core code modules, there are several auxiliary modules used to analyze, manipulate, and plot the dislocation objects.

In the remainder of this manual, we discuss how to use the three core modules using the example published in the manuscript above. The microscopy images from the manuscript is available with the ObDiTo source in the `test_data/medlin304L-testing` folder, so users can run the

[1] Available at http://micro.stanford.edu/~caiwei/Forum/2005-12-05-DDLab/.

analysis themselves to get a feel for how to use it. The results can be compared with the published results provided in the `test_data/medlin304L-archival` folder.

2. Necessary Resources

ObDiTo is written in MATLAB and requires the Image Processing Toolbox. It was written and tested using MATLAB 2020a and 2020b on a MacBook. It has not been tested on other platforms or other versions of MATLAB.

3. How to Use ObDiTo

3.1. Line Extraction

The `extract_lines` function has the following syntax:

```
extract_lines(path,fileName,mode,lseg,lmin)
```

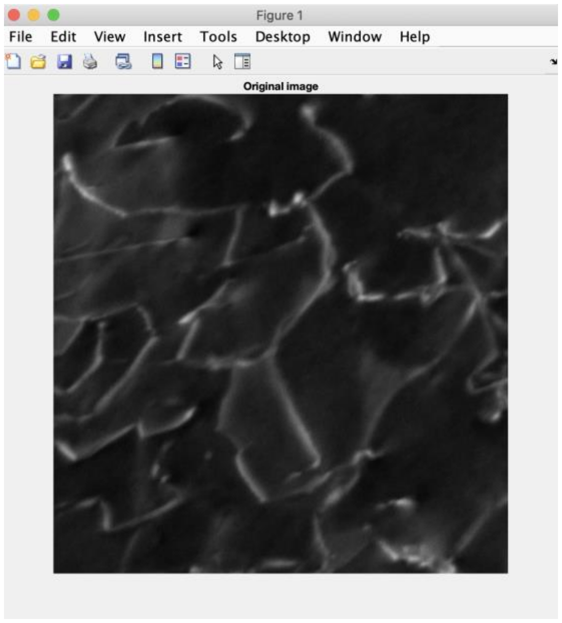
with the arguments

- `path` = folder path to the TEM image (string)
- `fileName` = name of the TEM image with extension (string)
- `mode` = string to specify how to run the function, valid choices are: `'new'` - use this option when loading an image for the first time, and `'mod'` - use this option when an image has already been analyzed once and the objective is to manually modify the line structure
- `lseg` = segment size to use in units of pixels (integer)
- `lmin` = minimum object length in units of pixels, any objects with length less than `lmin` are removed (integer)

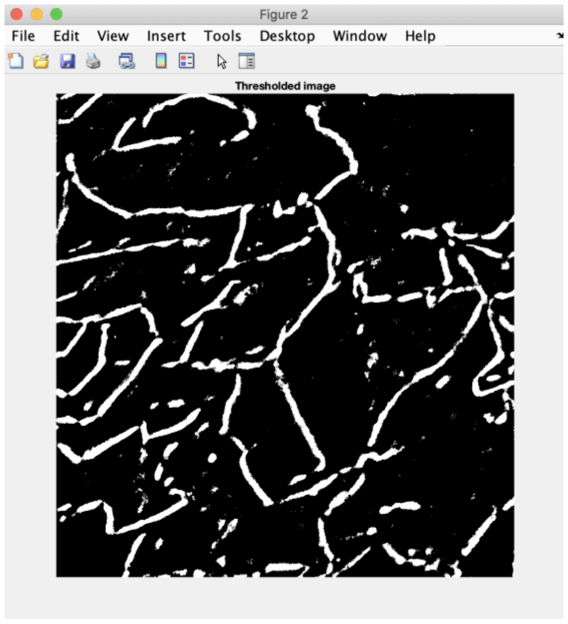
An example for how to call this function is:

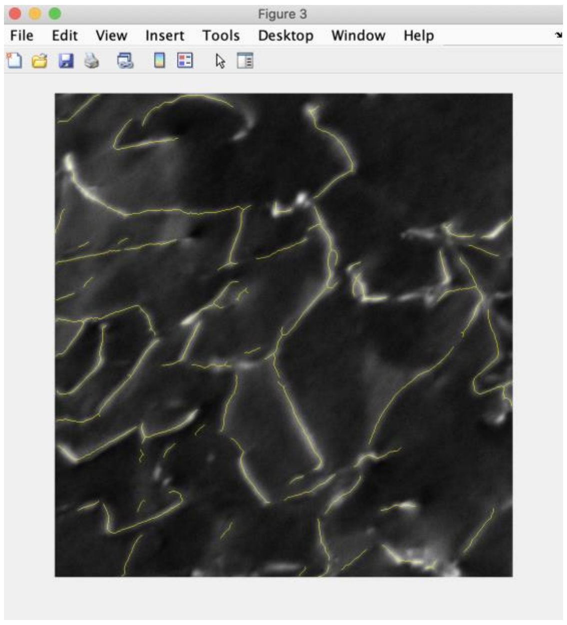
```
extract_lines('test_data/medlin304L-testing','frame01.tif',...
              'new',10,15)
```

Upon calling the function, the original image is displayed in Figure 1.



The code then attempts to automatically extract dislocation lines using thresholding and thinning. The thresholded image is displayed in Figure 2 and the resulting thinned image in Figure 3 superposed over the original image.

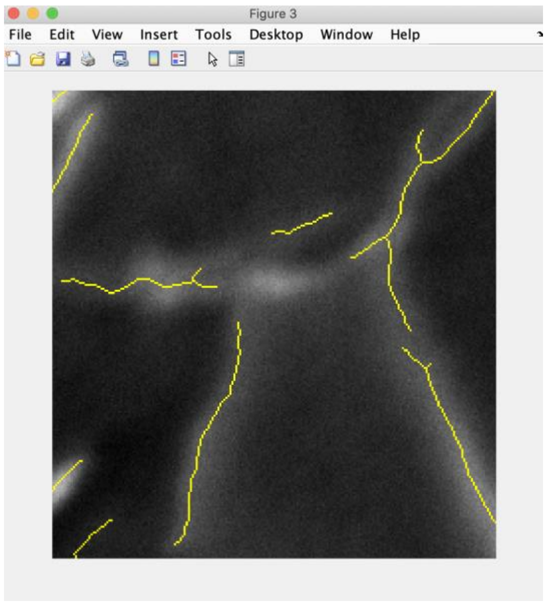


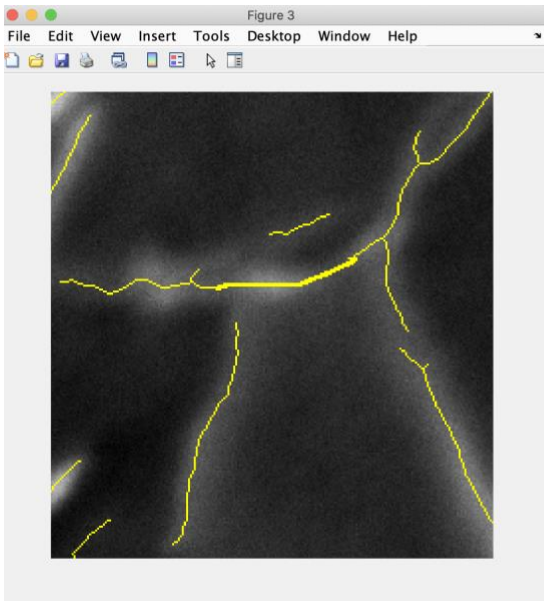


Clearly the automatic extraction process is imperfect, with incompletely extracted lines and misidentified lines. The user is next able to “clean up” the lines manually using an editing tool in Figure 3. The following prompt is given in the command window:

```
Enter l to draw a line, e to erase, u to undo, and q to quit:
```

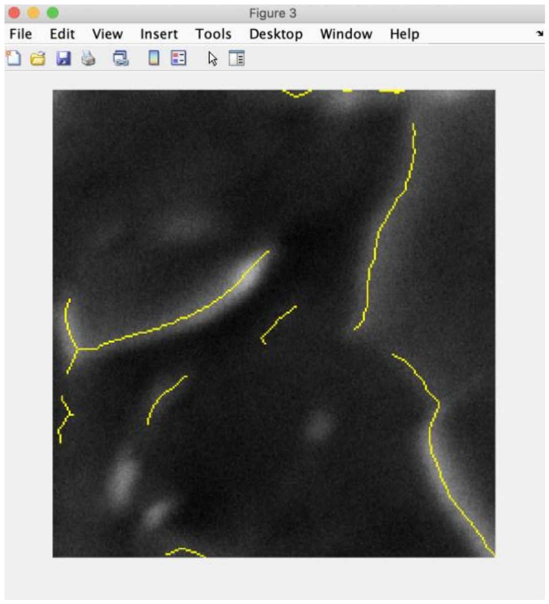
To draw a missing line or component of a line, type `l` in the command window and hit enter. You can then draw a line in Figure 3 by clicking and dragging on the screen. After releasing your mouse, the new line is automatically added, as shown below. The new line is visible on the right image.

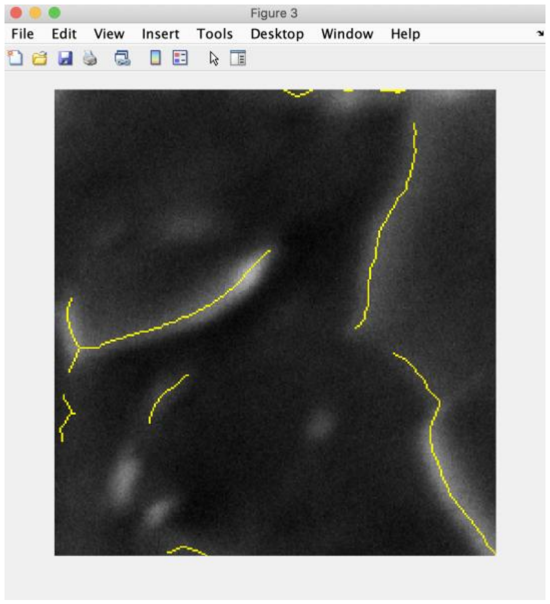




Note that the newly sketched line is slightly thicker than the other thinned lines. These will be thinned again at the end. You need to ensure that sketched lines are contiguous with pre-existing lines if they belong to the same dislocation (no pixel holes or gaps).

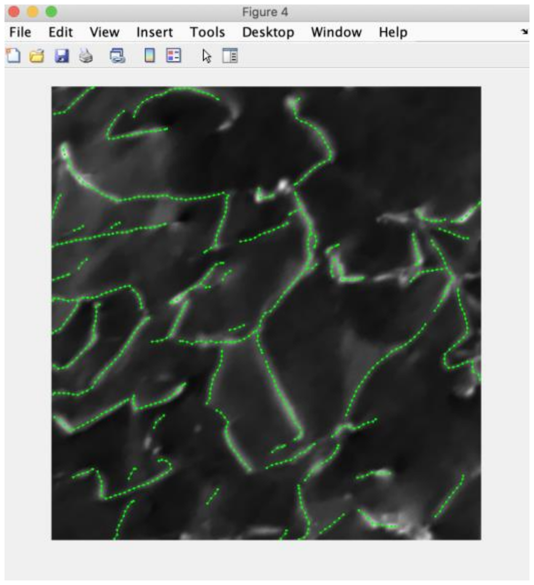
If instead you wish to erase an incorrectly identified line, type `e` in the command window and press enter. You can then select any lines to erase by encircling them in Figure 3. After releasing the mouse, anything that was encircled while clicking and dragging is erased, as shown below.





If you decide after making an edit that you are unhappy with the resulting changes, you can undo it by typing `u` in the command window and pressing enter. After you are satisfied with your final edits or need to end your editing session and plan to continue later, type `q` and press enter.

After quitting, another thinning pass is performed and the pixel skeleton is converted to the nodal-segment representation plotted in Figure 4, as shown below.



After quitting, two files will have been saved in the path specified in `extract_lines`:

`fileName_lines` – a binary image with the same extension as the original image (e.g., .tif in the example above), the contents of which is the skeleton obtained after manual editting

`fileName.mat` – a .mat file which stores the data structure for the dislocation objects in the form of rn and links arrays (see Section 1 for description, note that rn only contains [x,y] coordinates at this stage).

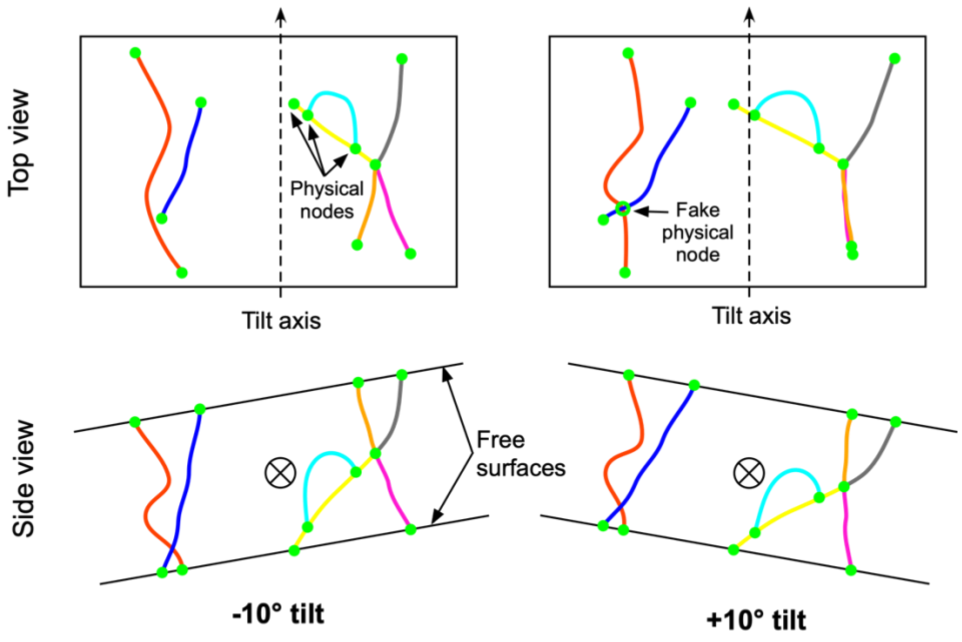
If you decide to continue editing your line structure, you can run the code using the `'mod'` mode, e.g.,

```
extract_lines('test_data/medlin304L-testing','frame01.tif',...
              'mod',10,15)
```

The code will automatically load the relevant `fileName_lines` file (if it exists) and you can continue editing. This process can be repeated as many times as is necessary, however, it does seem that the thinning operation can degrade the quality of existing thinned lines if applied over and over again (e.g., 10+ times). Hence, it is advised that the same file is not edited for times than necessary.

3.2. Object Matching

After extracting lines from more than one image, dislocation "objects" must be defined and matched across the images. We define dislocation objects as portions of dislocation lines which terminate at physical nodes, e.g., nodes that connect to more or less than two segments. In order to identify objects, the user must select pairs of physical nodes. However, when dislocations cross above and below each other in the TEM foil, they appear to intersect and form physical nodes, but in reality do not. We call these "fake physical nodes." See the figure below from Sills and Medlin (2022).



Such fake physical nodes cannot be used to define dislocation objects. Unfortunately, the only way that fake physical nodes can be identified at present is by careful inspection: a true physical node exists in all images and does not move along the tilt axis as the sample is tilted. Prior to matching dislocation objects, the user must identify fake physical nodes to ignore.

To perform object matching, the `match_objects` function has the following syntax:

```
match_objects(path,basename,files,mode)
```

with arguments

- `path =` folder path to the TEM image files (string)
- `baseName =` name for the image series, to be appended with "matches" and used for saving output (string)
- `files =` cell array containing a list of TEM image file names (no extension) (cell array of strings)

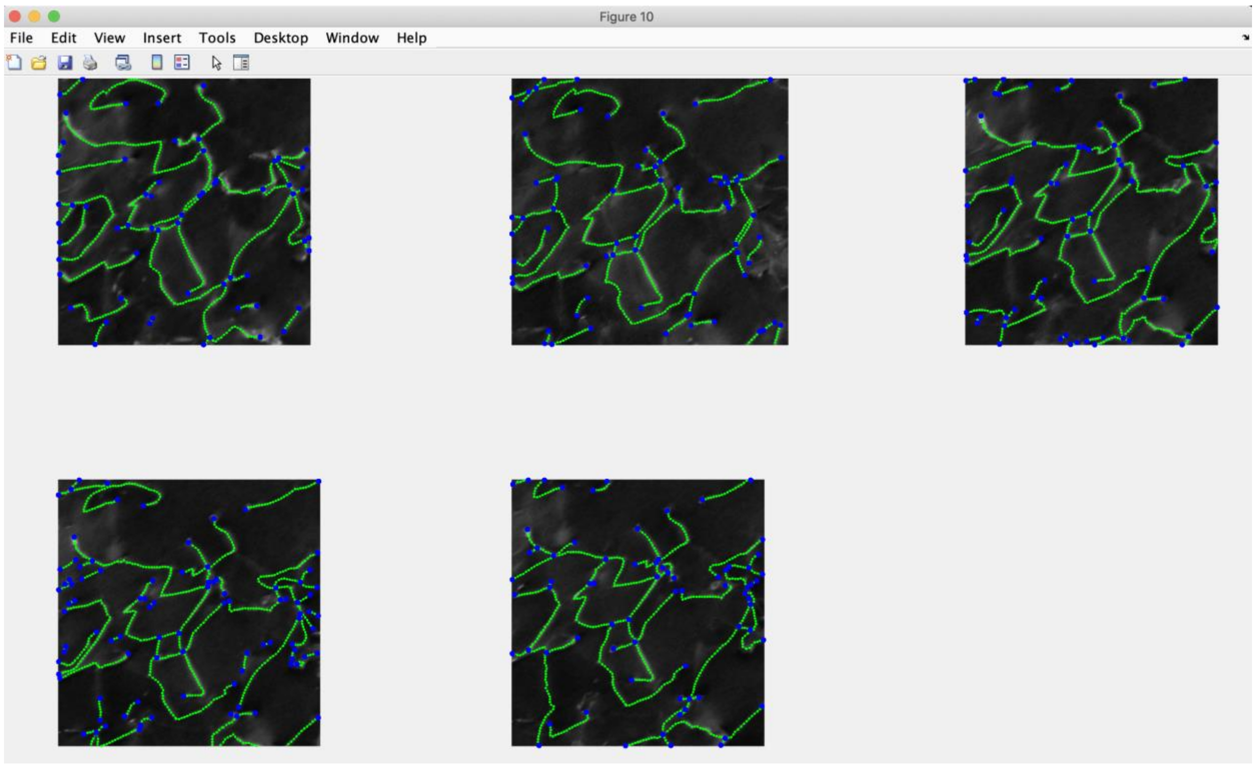
- `mode` = string specify how to run the function, valid choices are: `'new'` - use this option when matching objects for the first time, and `'mod'` - use this option when objects have already been matched for this dataset and you wish to continue matching

An example for how to call this function is:

```
match_objects('test_data/medlin304L-testing',...
              'medlin304_13456',{'frame01','frame03',...
              'frame04','frame05','frame06'},'new');
```

This assumes that you have already extracted lines from TEM images `frame01`, `frame03`, `frame04`, `frame05`, and `frame06`. To test the function you may copy the `frame*.mat` files from the `test_data/medlin304L-archival` folder into the `test_data/medlin304L-testing` folder rather than extracting lines from all images yourself. Note that you may also use fewer images, but must use at least two.

Upon calling the function, Figure 10 appears displaying all extracted dislocation lines:



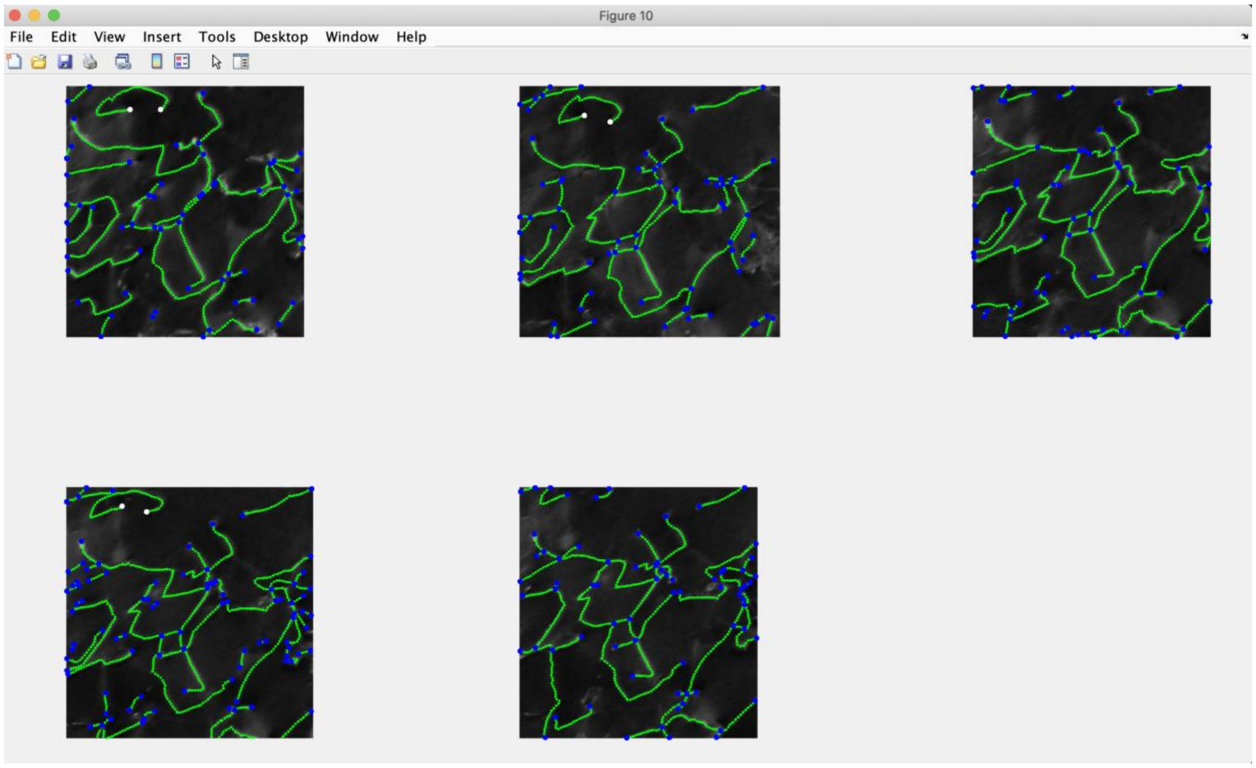
Potential (but possibly fake) physical nodes are displayed as blue. The command window prompt then states:

```
Press enter to continue selecting points, s to save, u to undo,
or enter q to quit:
```

The goal is to match the same dislocation object across several (ideally all) of the images. This is done by first selecting one physical node terminating that object in each image (moving from left to right, starting at the top), followed by the other physical node terminating that object. After pressing enter (with no other keystrokes) the command window displays:

```
Select the same physical node 1 on each image...
followed by the same physical node 2 on each image...
if there is no corresponding node in an image click out of the
image
```

To select physical nodes simply click near the node (the nearest blue node to the click is chosen). If you are skipping an image for a particular object, then click out of the image when it is that image's turn. After clicking as appropriate, the selected physical nodes are colored white:



In the example above, the "C-shaped" dislocation (top left) was not fully visible in images 3 and 5, and so was skipped by clicking out of the images.

Now continue this process for all dislocation objects of interest. If you make a mistake you can undo the last matching operation by entering `u` when prompted. You can also save at any point by entering `s` when prompted. To quit, enter `q` when prompted. After quitting, just the dislocation segments associate with selected dislocation objects are extracted from each image and displayed. These pared down datasets are then saved along with an array storing the object match information in the `basenamematches.mat` file in the specified `path` for use during the final reconstruction.

3.3. 3D Reconstruction

With dislocation objects matched, you are now ready to estimate the z height of each node. To perform the reconstruction, the `reconstruct_3dlines` function has the following syntax:

```
reconstruct_3dlines(path,basename,angles,ref,scale,do_plot)
```

with arguments

- `path` = folder path to the line and alignment data (string)
- `basename` = same basename as used in `match_objects` (string)
- `angles` = tilt angle for each of the images used with `match_objects` in degrees (array of doubles)
- `ref` = file number to use as a reference when performing the reconstruction (integer)
- `scale` = image scale in nm per pixel (double)
- `do_plot` = binary flag (0 or 1) indicating whether to plot the results

An example for how to call this function is:

```
[rn,links] = reconstruct_3dlines(...
                 'test_data/medlin304L-testing',...
                 'medlin304_13456',[-11.87 -2.38 2.38 7.11...
                                             11.87],2,0.409,1);
```

Note that it is necessary to have all of the image .mat files and the matches .mat file from the `test_data/medlin304L-archival` folder in the `'test_data/medlin304L-testing` folder in order to run this command.

After calling the function, no user input is necessary. If `do_plot = 1`, then results are plotted showing the recentering of the datasets (Figure 20), determination of the tilt axis orientation (Figure 21), determination of the z height for each node (Figure 22), and the final 3D reconstruction (Figure 23).

If users wish to manually plot and view the final reconstruction, they can utilize the `plotnodes` function as

```
plotnodes(rn,links,color)
```

where `color` is a MATLAB color string (e.g., `'r'` for red) chosen by the user.

Acknowledgements

ObDiTo was developed with support from Sandia National Laboratories under the Laboratory Directed Research and Development (LDRD) program.