Example Data for MATLAB image registration, counting, and nearest neighbor analysis.

Tutorial using example data:

1. KJ\_CoOEP\_O2\_80\_211102\_160950.bcrf
2. KJ\_CoOEP\_O2\_80\_211102\_161022.bcrf
3. KJ\_CoOEP\_O2\_80\_211102\_161054.bcrf
4. KJ\_CoOEP\_O2\_80\_211102\_161117.bcrf

These images have been plane corrected and Mean 3x3 (weak) filtered and saved as .bcrf files inside SPIP.

# STEP 1: Register and align images

Using register\_bcrStack.mlx

1. bcrf files above should be inside the same folder as the register\_bcrstack.mlx file
2. run the first section of code to load the images
3. run the second section of code to obtain the initial registration points in the first image.
   1. left click on the image to pick points, at least 3 but 4 preferred (see figure 1).
   2. points must appear in all images to be registered.



*Figure 1: Image 1 with the registration points shown.*

1. run the next code section. This section has a loop that will display every image and allow the user to choose points by left clicking on the image. Choose the same locations as image 1 in the same order.



*Figure 2: Example of registration points chosen for images 2-4 in the sequence.*

1. the next section will align the images based on the control point chosen in step 3 and 4.
2. the next section is for visualizing the registered result image one frame at a time.



*Figure 3: Resulting registered image 1.*

1. the final section will save the following registered image variables to a .mat file
   1. registered\_image\_full = array 279x539x4 double
      1. size 279x539 may be slightly different in your case it depends on the exact placement of the registration points.
   2. tform = 1x4 cell
      1. holds the 3x3 transformation matrix for each image
   3. mov = 4x2x4 double
      1. holds the 4x2 x,y coordinates of each registration point
      2. matlab arrays are indexed by (row index, column index, page index)
         1. to have matlab return the x coordinate of the 1st point in the 1st image use mov(1,1,1) = 386
         2. use mov(1,:,1)= [386, 51] to have matlab return the x and y coordinates – both columns for point 1 in image 1
         3. mov(:,1,1) returns all x coordinates for image 1= [386; 287; 188; 182].
         4. commas delineate row vectors and semi colons delineate column vectors.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Contents of mov variable. | | | | | |
|  | x | y |  | x | y |
| mov(1,:,1) | 386 | 51 | mov(1,:,3) | 382 | 53 |
| mov(2,:,1) | 287 | 193 | mov(2,:,3) | 287 | 194 |
| mov(3,:,1) | 188 | 163 | mov(3,:,3) | 116 | 164 |
| mov(4,:,1) | 182 | 68 | mov(4,:,3) | 181 | 70 |
|  |  |  |  |  |  |
| mov(1,:,3) | 383 | 57 | mov(1,:,4) | 377 | 57 |
| mov(2,:,3) | 285 | 198 | mov(2,:,4) | 281 | 198 |
| mov(3,:,3) | 112 | 168 | mov(3,:,4) | 110 | 167 |
| mov(4,:,3) | 179 | 71 | mov(4,:,4) | 176 | 72 |

* 1. im = 256x512x4 double
     1. original images as an array: 256(y length in pixels)x 512(x length in pixels) x number of images

1. After saving, you are ready to proceed with the image counting.

# STEP 2: Count all images

use sequentialImagesCounter1\_2.mlx

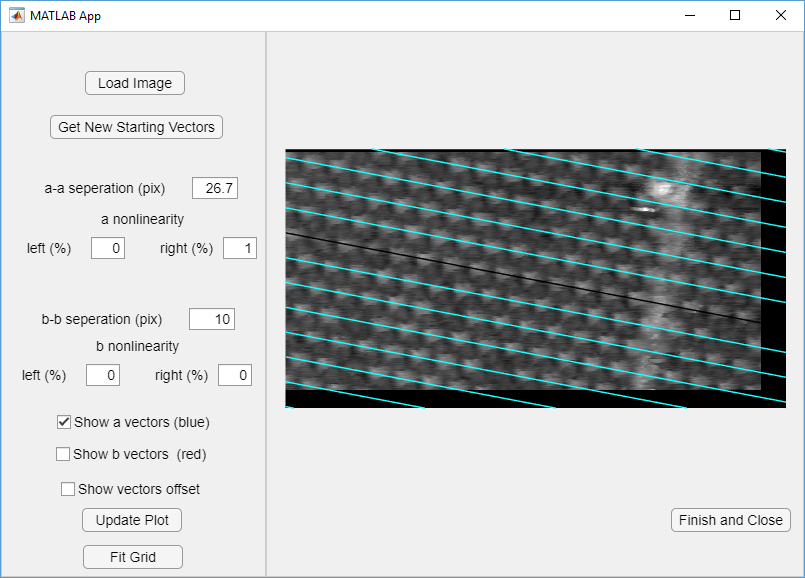
1. run section 1. This section clears the workspace of all variables and opens a dialog box where you can navigate to the .mat file that you just saved as part of the image registration step. This file’s data is loaded into matlab.
2. the next section runs the unitCellOverlay1\_4 app with the first image in the sequence. This app helps to fit a unit cell grid to the image. It prompts the user to left click on the image to choose points along any 1 unit cell vector then press enter then choose points along any other unit cell vector, Figure 4.



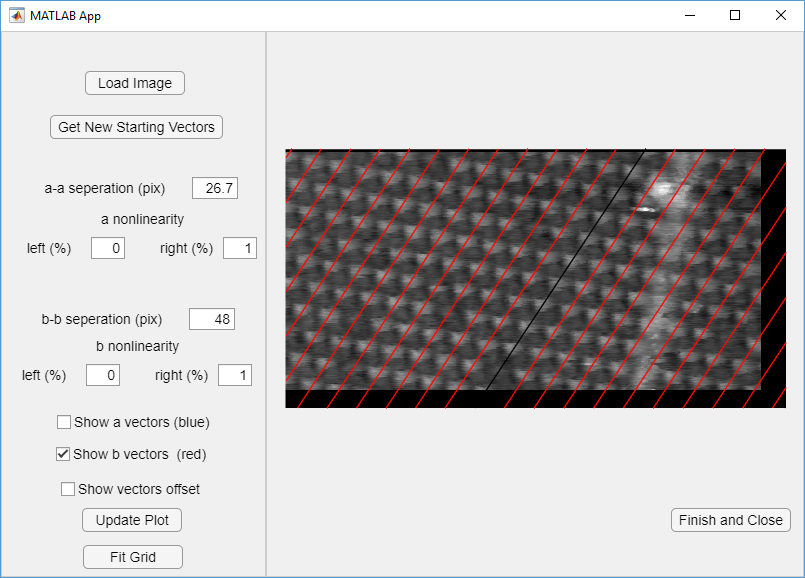


*Figure 4: Example of points chosen along the a and b unit cell vectors.*

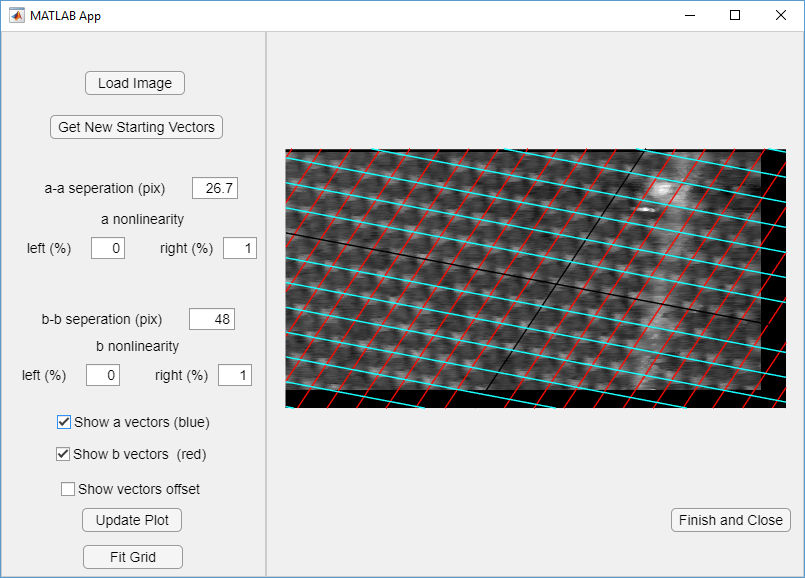
1. The next step is to adjust the a-a and b-b separation distance values in the app so that the a vectors (blue) and b vectors (red) fall along the centers of the molecules, see Figures 5 and 6.
   1. The nonlinearity parameters are used to help fit the separation between rows of molecules towards the edges of the image. Near the edges the distances may not match the separation distance near the image center if the image is distorted by drift or creep.
   2. the further the line is from the center black line, the greater the correction. Use positive number between 0 and 100 to increase the separation near the edges and a negative number between 0 and -100 to decrease the separation.



*Figure 5: a vectors with the correct separation distance and nonlinearity parameters.*

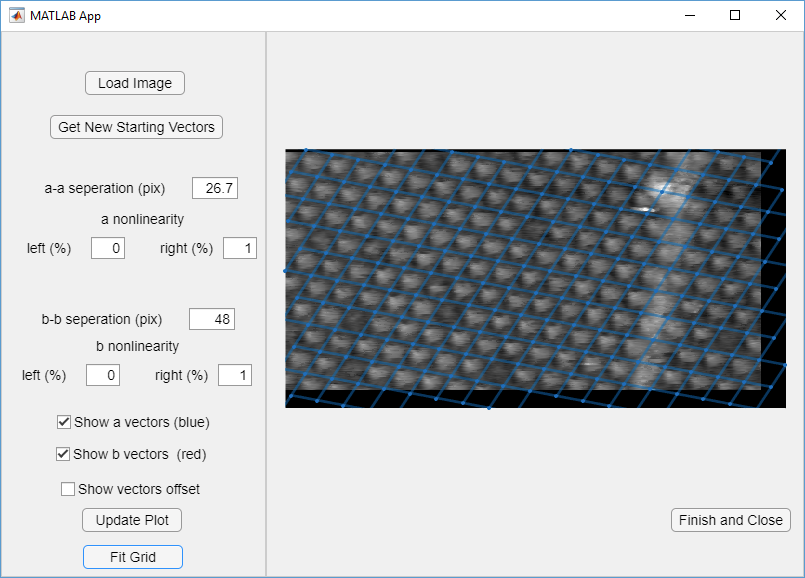


*Figure 6: b vectors with the correct separation distance and nonlinearity parameters*

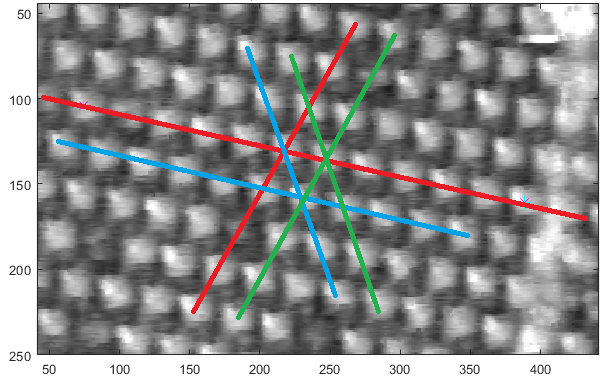


*Figure 7: screenshot of app showing both a and b vectors with the correct separation distances.*

1. After vectors are properly positioned then click the “fit grid” bottom at the bottom left side of the window and you will see a series of connected nodes and edges in a grid pattern, figure 8.
   1. Troubleshooting: sometimes the fitted grid does not match the image and if that happens the only way to fix it is to start over with different vector orientations. Click “get new starting vectors” button in top left then repeat step 2-4 until the grid matches the image, figure 9.



*Figure 8: Image after clicking “fit grid” button with correctly oriented grid.*



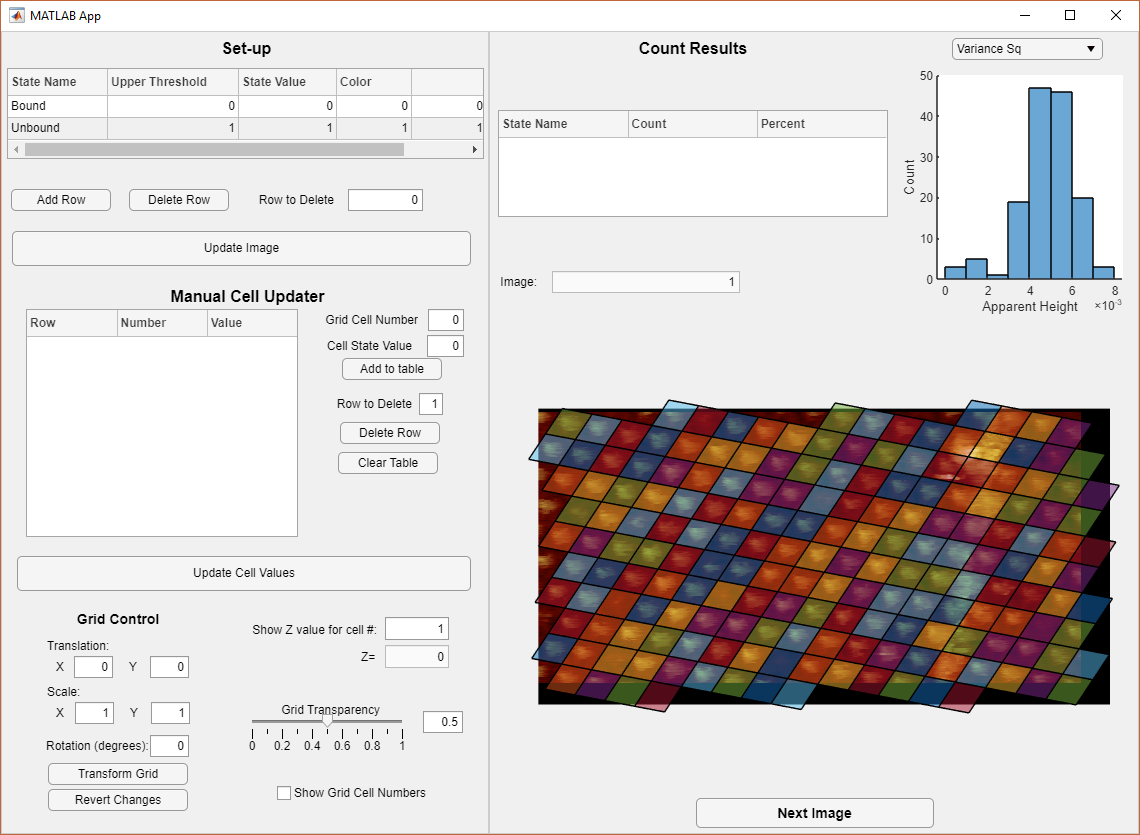
*Figure 9: Possible combinations of a and b vector choices. If points along the red line directions do not result in the correct final grid then choose points along the blue or the green directions and try to fit the grid again.*

1. Click finish and Close.
   1. this step closes the app and saves the required variables to the matlab workspace.
2. Run the next section to convert the node and edges seen in Figure 8. To a series of closed shapes. The closed shapes are necessary so that they can be color-coded based on the state of each molecule.
3. Then run the view grid section to double check the grid.
   1. this section has a few options associated with it, view\_opt, view\_num, and color\_code.
      1. view\_opt = which view option to use, 1 = graph - lines only as seen in figure 8, 2 = closed shape grid
      2. view\_num = option to view overlaid number for each grid cell 1= on, 0=off.
      3. color\_code = option to view color coded grid based on molecule state. 1 = on, 0 = off.
         1. only works after running the threshold counter app



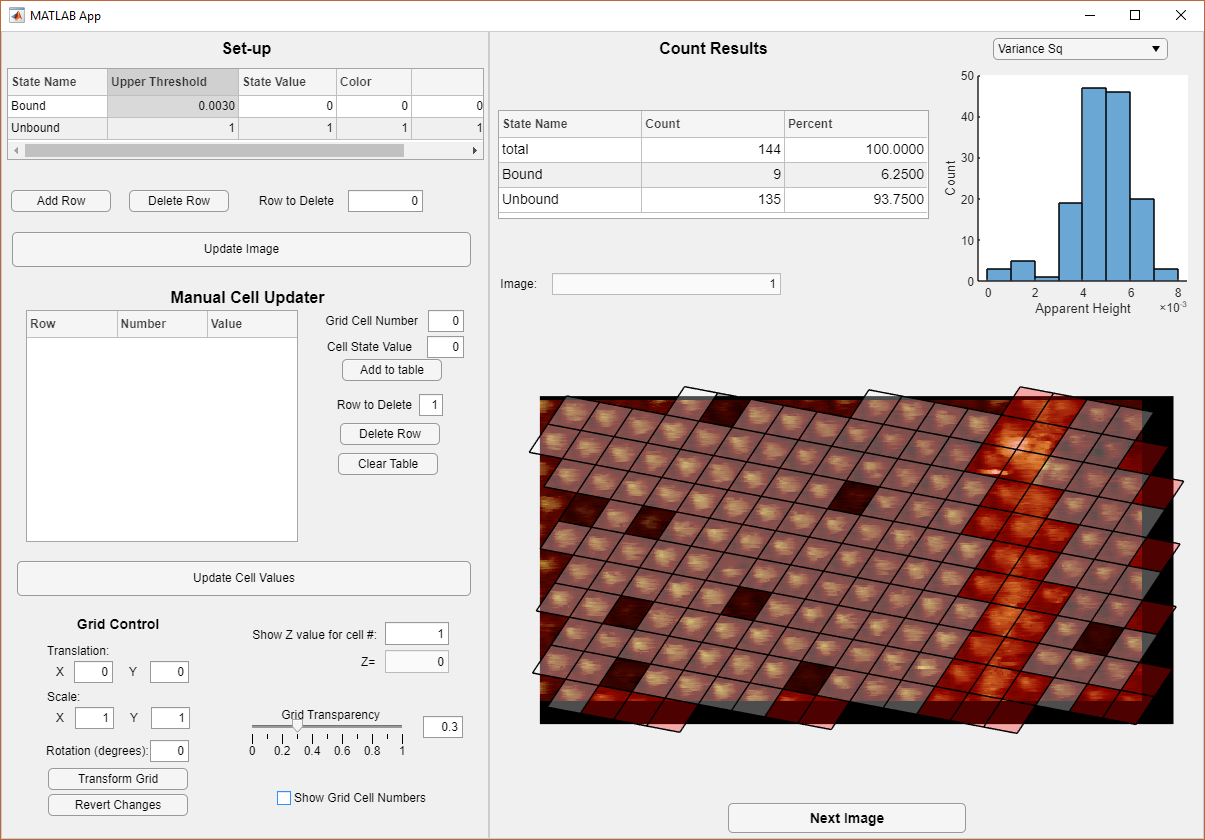
*Figure 10: Image with overlaid closed shape grid. Left options = view\_opt=2, view\_num = 0. Right options = view\_opt=2, view\_num=1.*

1. Set marker\_nums value. Marker\_nums takes grid cell numbers of any defect marker in the image so that those cells are not counted towards the total number of molecules in the image.
   1. for image shown in figure 10, marker\_nums = [1 2 9 10 21 182 37 38 67 53 70 69 172 96 83 98 99 114 115 166 127 128 146 155 147];
      1. numbers can be separated by commas or spaces in the list
   2. if the image did not contain a marker the marker\_nums can be empty. marker\_nums = [];
2. Now you are ready to count each image using the imageThresholdCounter1\_4.mlapp. This can be ran from Stage 3: Counting portion of the sequentialImagesCounter. Run this code section and the counter app will launch. The app will initially look like figure 11.



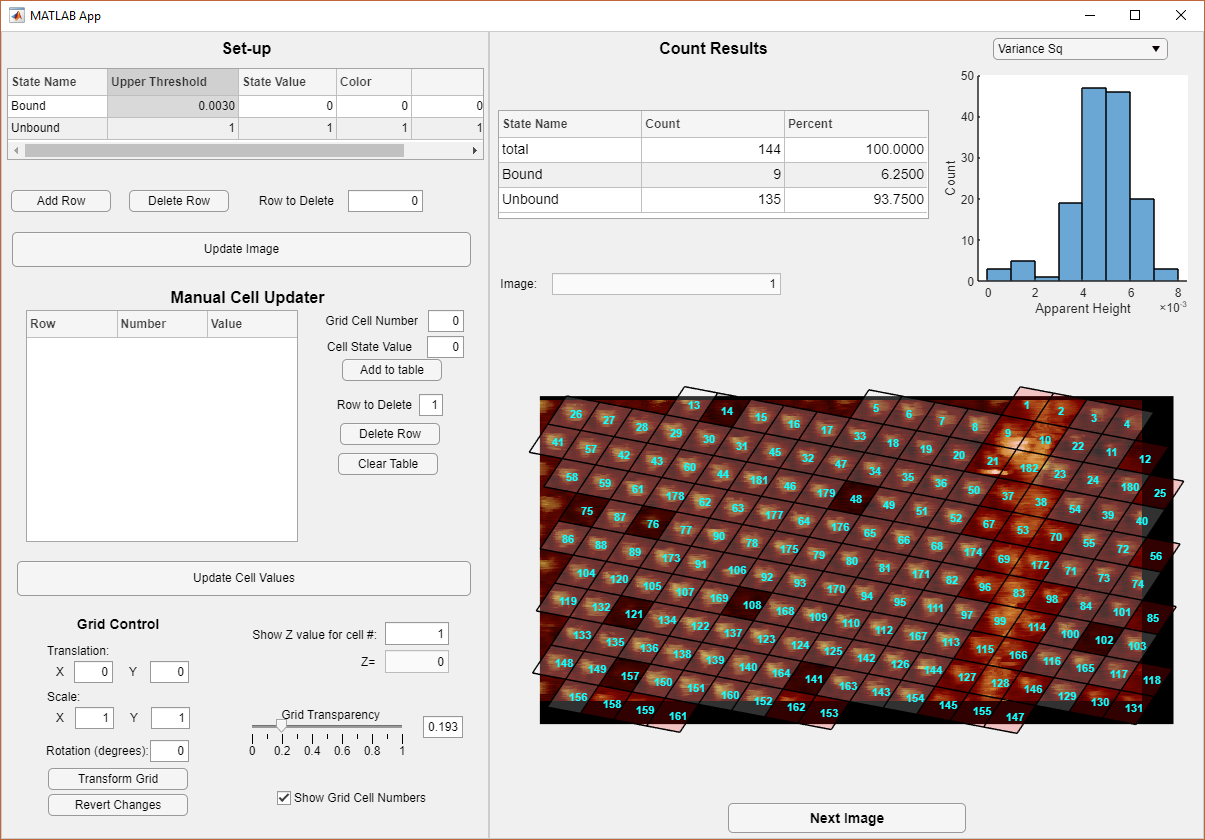
*Figure 11: Initial view of counter app.*

1. The next step is to set a threshold value based upon the height histogram in the top right corner. By default the histogram is populated by the square variance of the apparent height inside each grid cell. variance2 = mean( (apparent height - mean(apparent height))2 )
   1. For this example I choose a upper threshold for the bound molecular state to be 0.003 and I entered this number into the set-up table in the top left. Then click update image. Result is shown in figure 12.



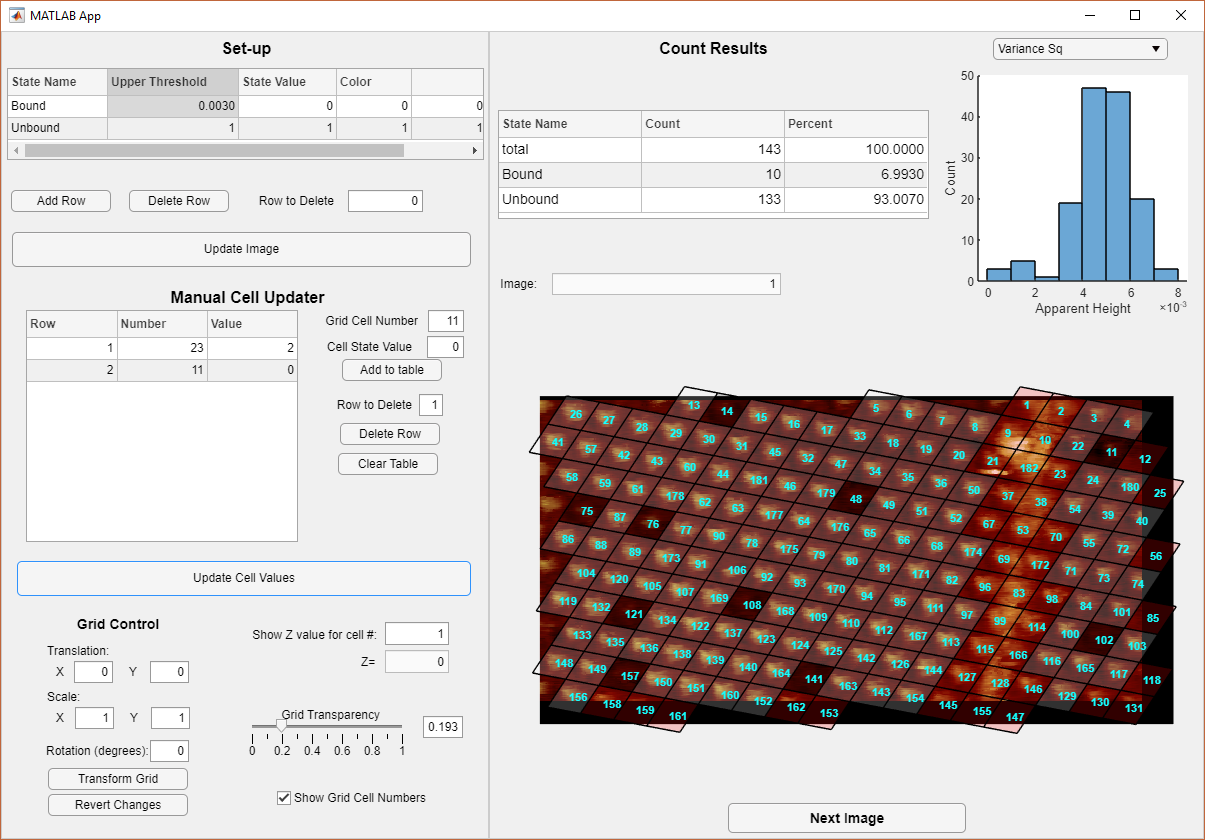
*Figure 12: Result after choosing 0.003 as an upper threshold and clicking “update image” button.*

1. From here one can manually update any missed molecules. Near the bottom left click the “show grid cell numbers” checkbox to view the overlaid cell numbers. The transparency level grid of the grid can be adjusted with the slider. The show z-value for cell #: box can be used to display the height value for any individual cell and may help in choosing the threshold value.
   1. black = bound and has state value = 0
   2. white = unbound and has state value = 1
   3. red = not counted and has state value = 2



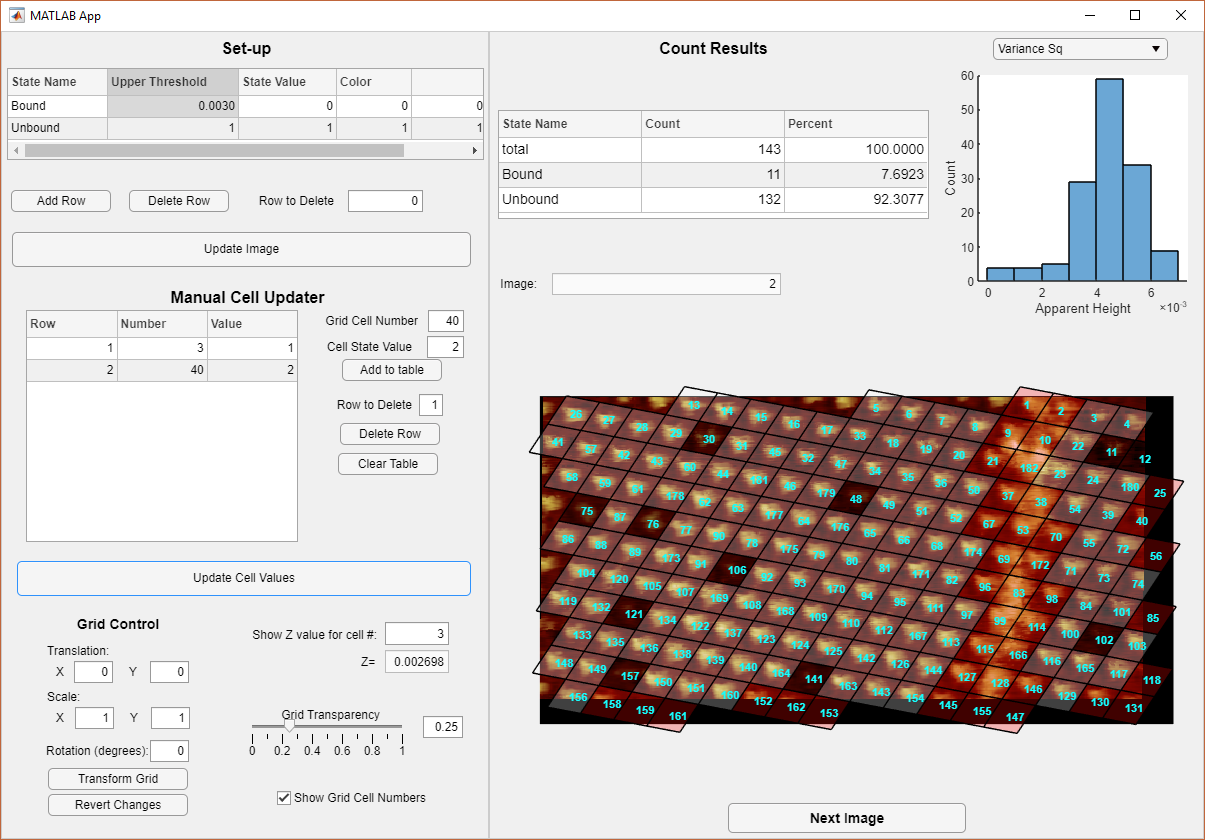
*Figure 12: Example image with grid cell numbers on. In this case, molecule 23 should not be counted because it is close to the marker and molecule 11 should be counted as bound.*

1. To manually update the cell values use the Manual Cell updater section on the left side of the app. Any grid cell number that must be updated is input into the “grid cell number” text box and the new state value will be input into the “cell state value” text box. Then click “add to table” button. In this example, molecules 23 and 11 need to be manually updated. 23 should not be counted and its new state value should be 2 and 11 should be counted as bound with a new state value of 0. Once the table contains all of the cells that need to be updated then click “Update cell values” button. See figure 13 for resulting app display. Notice that the count Results table in the right side of the app is automatically updated to reflect the changes.

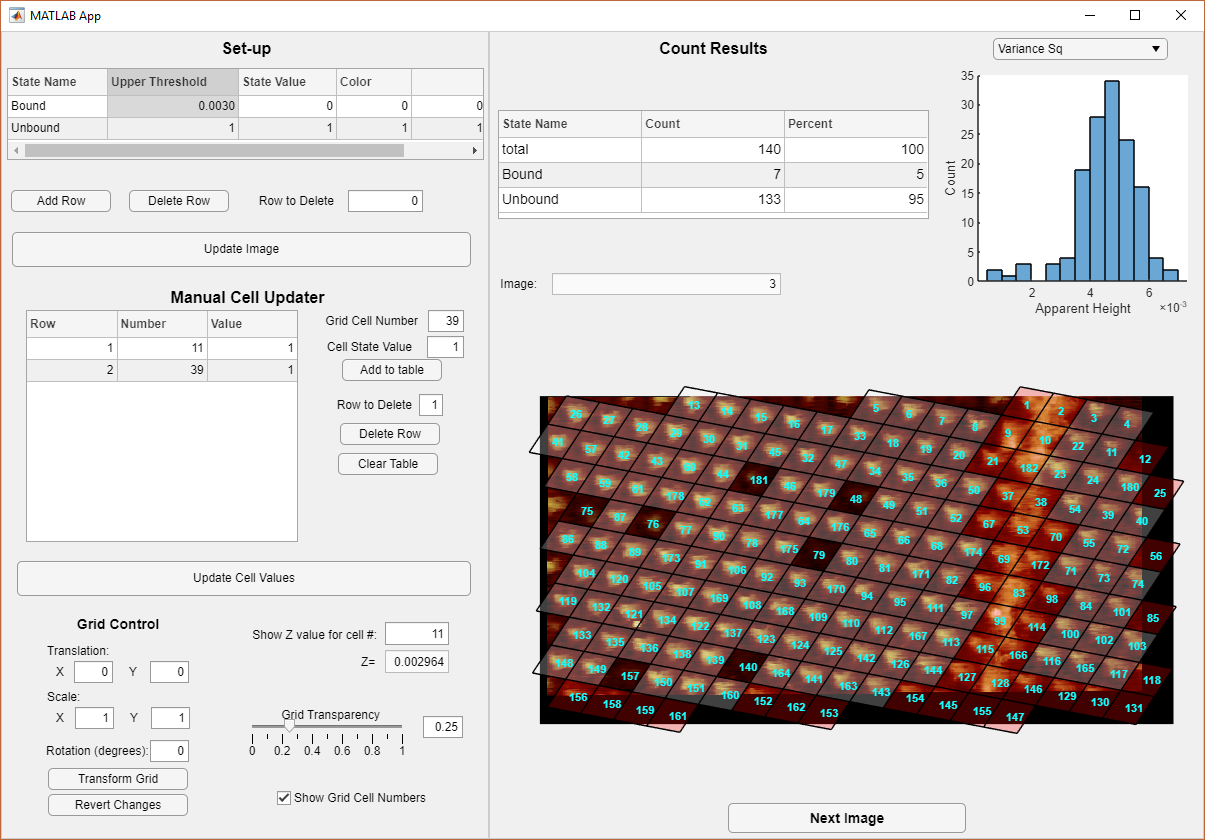


*Figure 13: Display after manually updating cells 23 and 11 and clicking “Update Cell Values” button.*

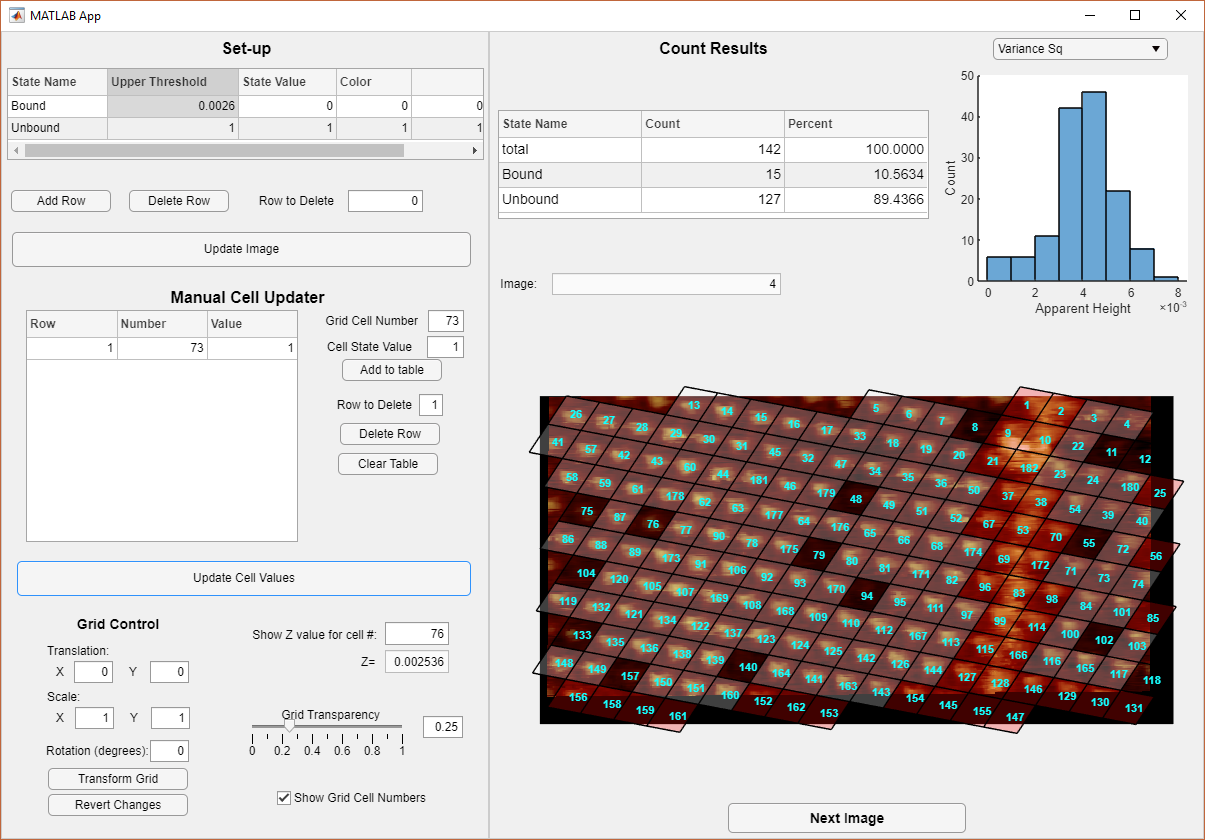
1. When finished with image 1 then click “next image” button to proceed. Repeat the steps with each image in the sequence. Remember to click “update image” button if the threshold value is changes. See figures 14-16 for counting result of each image. Once the end of the sequence is reached then pressing “next image” button will close the app and save the results to the matlab workspace.



*Figure 14: Image 2 counting result.*



*Figure 14: Image 3 counting result*



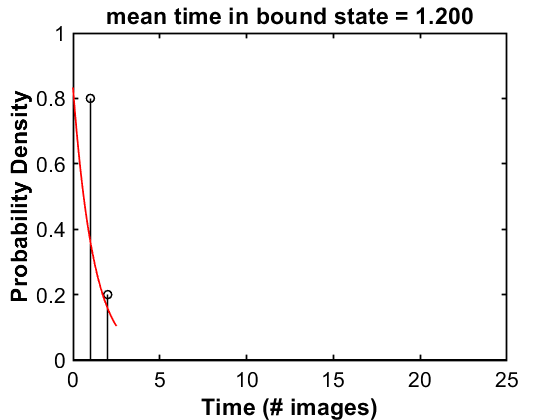
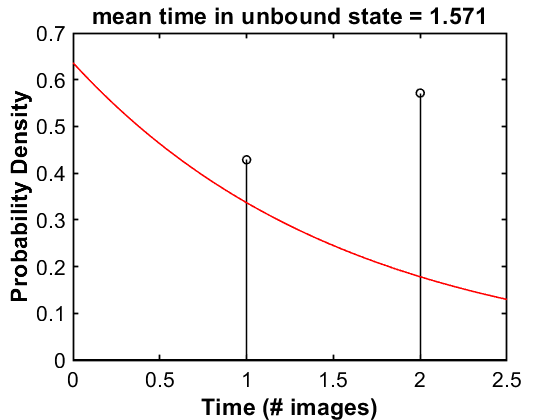
*Figure 15: Image 4 counting result.*

1. Immediately run the first save results section in order to save the results to the permanent matlab variables. If there was an error in any image in the sequence, the counter app can be restarted with only that particular image. In the previous code section in the sequential images counter script the im\_range variable can be updated to have im\_range = [# start image, # end image]. If you would like to redo image 3 then im\_range = [3, 3]. Use the prior view grid code section to check the images.
2. Next run the summary table section to output a summary of the results.

| **Name** | **Number of Images** | **mean**  **molecules/image** | **sum molecules in all images** | **mean bound**  **molecule/image** | **sum bound molecules in all images** | | **mean coverage** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 'example 1' | 4 | 142 | 568 | 10.7500 | 43 | 0.0756 | |

# STEP 3: Dwell time analysis

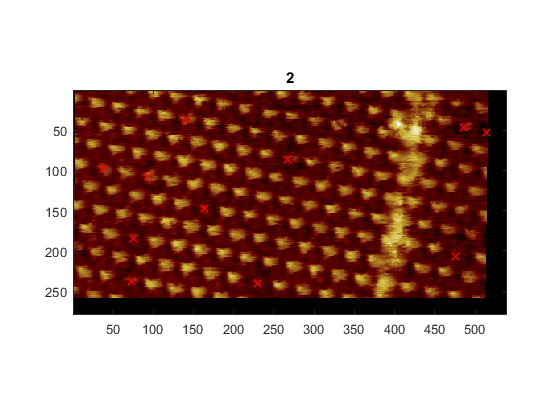
1. Then the dwell time analysis can be completed by running the dwell time calculation section. This section will also create histograms of the dwell times and fit them with exponential probability distributions.



*Figure 16: Result of dwell time analysis for example images.*

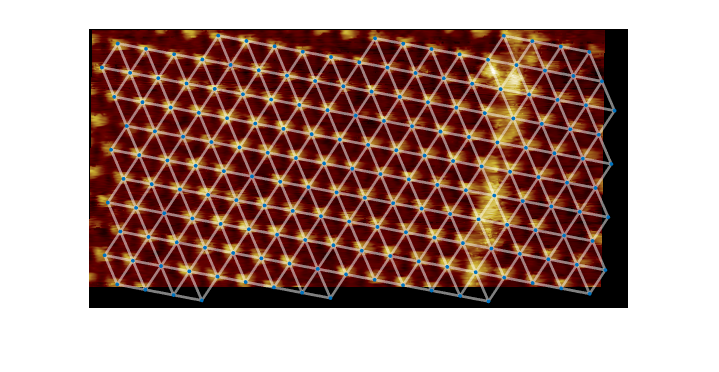
# Step 4: Nearest Neighbor Analysis

1. Finally, the number of nearest neighbor molecules can be counted. Using the nearest neighbor analysis section. This section should ran once for each image in the sequence. Enter the image number that you want to look at in the nim variable. Edge molecules that do not have all 6 nearest neighbors are displayed with red xs and are not included in the total number of bound molecule counts, but are considered possible neighbors of inner molecules.
   1. Troubleshooting: Sometimes the grid does not work as planned and if that happens you will need to manually adjust the search range inside the calc\_nn function. This is an embedded function inside the sequential counter script included at the end of the file. You may see that the neighbors are not picked up, see Figure 17.

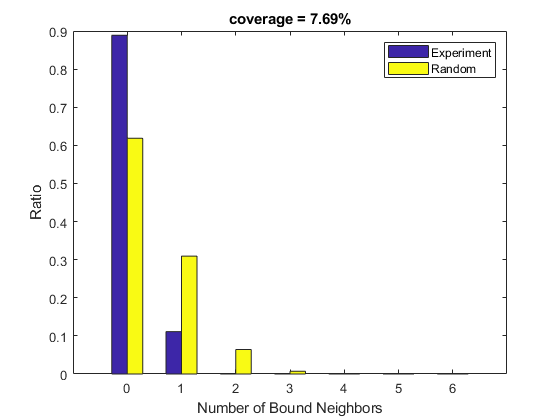
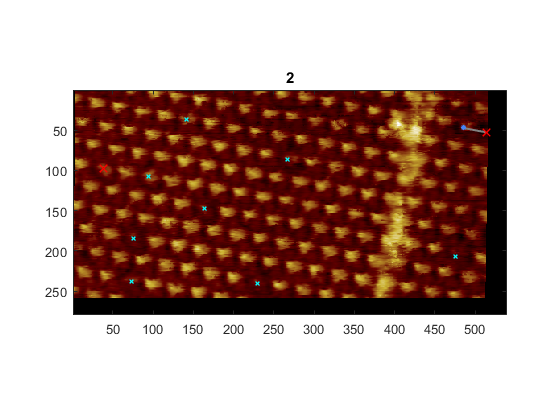


*Figure 17: Example of neighbors that are not counted in top right corner.*

* 1. If neighbors are not found then run the next code section for help troubleshooting. This section will output a length in pixels, ed\_6 and an image that should look like Figure 18 where each molecule is connected with 6 neighboring molecules. In this case ed\_6 = 39.81. calc\_nn function should be updated with this value rounded down.
     1. line 382: [G\_edge, ed\_len] = fitGraph(mpts, 6, XX); %XX=ed\_6
     2. line 382: [G\_edge, ed\_len] = fitGraph(mpts, 6, 39);
  2. After updating the line of code then rerun the nearest neighbor section.



*Figure 18: Display result of troubleshooting section. Each molecule/node should have 6 connections leaving the node.*



*Figure 19: Nearest neighbor display after adjusting the search range.*

1. Last of all the counts for the nearest neighbor analysis and the general counting can be output to an excel file. Update the excelName variable with your desired filename for the excel file. excelName can be anything but must end with .xlsx.
   1. e.g. excelName = ‘example1.xlsx’
2. Counting Complete!!

# Other things of note:

.mat result files can be loaded into matlab by dragging the file into the command window section of matlab or by double clicking the file name in the current folder section of matlab on the left hand side.